PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : H04L 12/58, H04M 3/533 — A1

(11) International Publication Number: WO 00/31930

(43) International Publication Date: 2 June 2000 (02.06.00)

(21) International Application Number: PCT/GB99/03831

(22) International Filing Date: 17 November 1999 (17.11.99)

(30) Priority Data:
98309679.3 25 November 1998 (25.11.98) EP

(71) Applicant *(for all designated States except US)*: BRITISH TELECOMMUNICATIONS PUBLIC LIMITED COMPANY [GB/GB]; 81 Newgate Street, London EC1A 7AJ (GB).

(72) Inventors; and

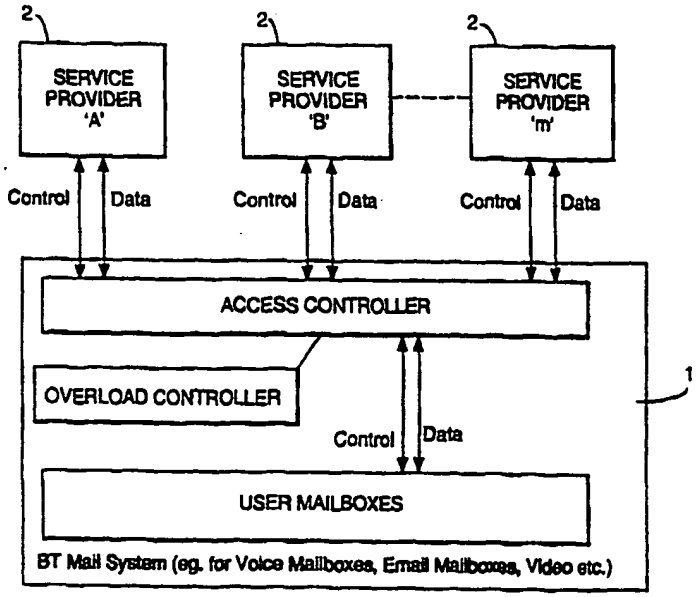
(75) Inventors/Applicants *(for US only)*: BALE, Melvyn, Christopher [GB/GB]; 23 Digby Road, Ipswich, Suffolk IP4 3NB (GB). HENDERSON, Gordon [GB/GB]; 51 The Lloyds, Kesgrave, Suffolk IP5 2WH (GB).

(74) Agent: WELLS, David; BT Group Legal Services, Intellectual Property Dept., 8th Floor, Holborn Centre, 120 Holborn, London EC1N 2TE (GB).

(81) Designated States: AU, CA, JP, SG, US, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: MESSAGING PLATFORM



(57) Abstract

A messaging platform, for example a platform storing email or voicemail, is provided with an overload controller which detects an overload condition and then acts to limit the loading of the platform. The overload controller may be associated with a control interface used, for example, for communication between a service provider and the message platform, and may selectively deny some of the requests received over that interface when the overlaod condition is detected.

PCT GB99/03831
29 FEBRUARY 2000

Europäisches Patentamt | European Patent Office | Office européen des brevets

EPO

GB99/3831

Bescheinigung | Certificate | Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. | Patent application No. | Demande de brevet n°

98309679.3

Der Präsident des Europäischen Patentamts:
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

I.L.C. HATTEN-HECKMAN

DEN HAAG, DEN
THE HAGUE, 24/02/00
LA HAYE, LE

Europäisches Patentamt | European Patent Office | Office européen des brevets

Blatt 2 der Bescheinigung
Sheet 2 of the certificate
Page 2 de l'attestation

Anmeldung Nr.
Application no
Demande n°
98309679.3

Anmeldetag
Date of filing
Date de dépôt
25/11/98

Anmelder
Applicant(s)
Demandeur(s)
BRITISH TELECOMMUNICATIONS public limited company
London EC1A 7AJ
UNITED KINGDOM

Bezeichnung der Erfindung
Title of the invention
Titre de l'invention
Messaging platform

In Anspruch genommene Priorität(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat State Pays	Tag Date Date	Aktenzeichen File no Numéro de dépôt

Internationale Patentklassifikation
International Patent classification
Classification internationale des brevets
H04M3/50, H04L12/58

Am Anmeldetag benannte Vertragstaaten
Contracting states designated at date of filing AT/BE/CH/CY/DE/DK/ES/FI/FR/GB/GR/IE/IT/LI/LU/MC/NL/PT/SE
Etats contractants désignés lors du dépôt

Bemerkungen
Remarks
Remarques

MESSAGING PLATFORM

The present invention relates to a messaging platform suitable for use in a communications network.

Users of communications networks are now commonly offered a variety of messaging services. By contrast with conventional telephony services in which calls are established in real time, in messaging services a message, which may be in any of a number of media, is deposited at one time and read at a later time. These services may include, for example, voicemail allowing the user to record a spoken message, faxmail that similarly allows a fax to be stored for later retrieval by the addressee, or email

According to a first aspect of the present invention, there is provided a messaging platform including

a) a message store arranged to receive message data and to store said message data for subsequent retrieval,

characterised by

b) an overload controller responsive to an overload condition in the platform and arranged, in response to the said overload condition, to limit loading of the platform

While it is known to provide platforms for real-time telephony with overload control mechanisms, conventionally messaging platforms have been regarded as carrying out an "off-line" data storage and processing function and so have not included such overload control mechanisms. The present inventors have found however that the addition of an overload controller to a messaging platform significantly enhances the stability and scaleability of the platform, that is to say its ability to adapt to increasing numbers of users.

The term "overload condition" as used in this document encompasses conditions in which the loading of the platform is approaching a level at which the operation of the platform would be impaired, as well as conditions in which such a level has already been reached. In general the overload controller will be arranged to limit the loading of the platform is such a way that the level at which impairment occurs is not reached.

Preferably the platform further comprises

c) a control interface arranged to receive control requests instructing transactions on the messaging platform, and the overload controller includes means for denying at least some of the control requests in response to the overload condition.

While at one time the messaging platforms supporting such services were operated solely under the control of the network operator, increasingly messaging services are operated by service providers other than the network operator. Accordingly it is necessary to provide the messaging platform with a control interface to allow communication of control signals with the service provider. In addition, in order to implement more advanced services which go beyond mere message deposit and retrieval, it necessary to provide users with some access to a control interface. The present inventors have recognised that overloading of the control interfaces on a messaging platform is a critical factor in degrading the reliability of such platforms. This preferred feature of the invention, by providing an overload controller on the control interface significantly enhances the performance of the platform and makes it possible to open control interfaces to service providers or to users without compromising the stability of the platform.

Preferably the overload controller includes a store programmed with criteria for applying different classes of service to (a) the mailboxes in the mail store, (b) the service providers accessing the mail store, (c) control messages at the control interface, or any combination of these. The overload controller is arranged, in response to an overload condition on the platform or the service provider, selectively to deny access to a control message depending on a class of service assigned in accordance with the said criteria to the control message.

Preferably the messaging platform includes a plurality of mailboxes containing message data, each mailbox being switchable between an open state, in which message data may be written to or read from the mailbox, and a closed state, and the overload controller is arranged to allow requests for the closing of a mailbox and to deny requests for the opening of the mailbox.

The inventors have found that a particularly effective means of limiting the loading of the platform, while also limiting the impact on ongoing transactions, is to admit requests for closing mailboxes and to deny requests for opening mailboxes. Since requests for opening mailboxes are normally followed by a number of related transactions, denying these requests, while allowing existing

transaction sequences to be terminated by the closing of a mailbox, quickly results in a reduction in the loading of the platform.

According to a second aspect of the present invention, there is provided a messaging system comprising:

a service platform running a messaging service application; and

a messaging platform in accordance with the first aspect connected to the service platform via a control interface, and arranged to receive control requests from the service platform via the control interface.

The service platform may be integrated with the messaging platform, but preferably is remote from it. Several such service platforms may communicate with a single messaging platform.

According to a third aspect of the present invention, there is provided a method of operating a messaging platform including:

a) storing message data on the messaging platform:

b) subsequently outputting message data from the platform, thereby allowing retrieval of a corresponding message;

characterised by

c) detecting an overload condition on the platform and in response to the overload condition limiting loading of the platform.

Systems embodying the present invention will now be described in further detail, by way of example only, with reference to the accompanying drawings, in which;

Figure 1 is a schematic of a network incorporating a messaging platform;

Figure 2 is a schematic of a platform embodying the invention for use in the network of Figure 1;

Figures 3a to 3y are SDL diagrams showing in further detail the platform of Figure 2; and

Figure 4 is a diagram showing in further detail the platform of Figure 2.

As shown in Figure 1, a communications system includes a messaging platform 1. The messaging platform 1 is connected to a service provider platform 2. The service provider platform is connected to a number of communications networks including an IP (internet protocol) network 3 and the PSTN (public switched telephone network) 4. Customer terminals 5,6,7 are connected to the service provider platform variously via the PSTN and via the IP network. The

customer terminals may include for example, conventional fixed-line telephones, mobile telephones and also personal computers.

In use, customers may subscribe to a message service offered by the service provider. Then, for example, when a subscriber is not able to take a call from a calling party, the call may be diverted to the service provider platform. The service provider platform may then communicate data received from the caller, for example in the form of a recorded digitised voice message, to the messaging platform.

Figure 2 shows in further detail the messaging platform. This may be implemented, for example, using the BT CallMinder platform (described in [British Telecom Engineering Vol 15 Part 2 July 1996]), which is based on the Speech Applications Platform (described in [British Telecom Engineering Vol 13 Part 4 January 1995]). The CallMinder platform is an integrated platform implementing both a service application platform and the messaging platform. Alternatively a distributed architecture may be used in which a number of service provider platforms communicate with a message platform over a network. In this case, a suitable architecture is that described in Messaging APIs for Voice Networks, RM Claxton, Sixth IEE Conference on Telecommunications (no. 451) 1998.

As shown in Figure 2, a number of service providers, A,B,....m are connected to the messaging platform by control and data channels. As noted in the introduction above, some end users may themselves be given access to the control interface of the messaging platform, and the term "service provider" is used here broadly to encompass any party accessing the control interface, including such end users. The control channels are used for registration commands, for example to set event detection triggers on the platform to generate a notification whenever a message is left for a specified subscriber, and for action commands, for example to carry out actions such as replaying a message, saving a message or deleting a message. Both data and control channels from the service providers pass in the first instance to an access controller. The access controller is connected to an overload controller. The overload controller and access controller in combination function to protect the platform from overloading by signals arriving on the service provider interface. The overload controller detects overload conditions, for example by monitoring the rate of transactions between the access controller and service providers or alternatively by monitoring the processor

utilisation of the mail store processors. When an overload condition is detected by the overload controller then the overload controller configures the access controller to deny access to the platform to certain signals (for example, requests to open a low priority mailbox in the mail store). Different types of signals are assigned different priorities. Three alternative schemes are described below by way of example.

Scheme 1

In this scheme, different service classes are applied to different service providers. During overload conditions, as monitored by a load controller, those subscribing to a higher priority class of service level agreement are given priority over those requests coming from a Service Provider with a lower priority class. Hence, actions pertaining to the particular mailboxes used by customers of the higher priority class of service would receive preferential treatment. It is envisaged that a range of service classes will be defined, which will correspond to the overload mechanism achieving a specified quality of service for those classes. Examples of service classes are given in Table 1 below.

Service Class	***Description***
III	Guaranteed end user access 99.99% of the time Guaranteed Service Provider notifications within 30 seconds, 99.99% of the time
II	Guaranteed end user access 98.0 % of the time Guaranteed Service Provider notifications within 30 seconds, 90.0% of the time Guaranteed Service Provider notifications within 20 minutes, 99.99% of the time
I	No guaranteed end user access No guaranteed Service Provider notifications

Scheme 2

As in scheme 1 different service classes are used. In this case the different service classes are applied to individual mailboxes for particular subscribers. Mailbox subscribers subscribing to higher service level classes are given priority

utilisation of the mail store processors. When an overload condition is detected by the overload controller then the overload controller configures the access controller to deny access to the platform to certain signals (for example, requests to open a low priority mailbox in the mail store). Different types of signals are assigned different priorities. Three alternative schemes are described below by way of example.

Scheme 1

In this scheme, different service classes are applied to different service providers. During overload conditions, as monitored by a load controller, those subscribing to a higher priority class of service level agreement are given priority over those requests coming from a Service Provider with a lower priority class. Hence, actions pertaining to the particular mailboxes used by customers of the higher priority class of service would receive preferential treatment. It is envisaged that a range of service classes will be defined, which will correspond to the overload mechanism achieving a specified quality of service for those classes. Examples of service classes are given in Table 1 below.

Service Class	***Description***
III	Guaranteed end user access 99.99% of the time Guaranteed Service Provider notifications within 30 seconds, 99.99% of the time
II	Guaranteed end user access 98.0 % of the time Guaranteed Service Provider notifications within 30 seconds, 90.0% of the time Guaranteed Service Provider notifications within 20 minutes, 99.99% of the time
I	No guaranteed end user access No guaranteed Service Provider notifications

Scheme 2

As in scheme 1 different service classes are used. In this case the different service classes are applied to individual mailboxes for particular subscribers. Mailbox subscribers subscribing to higher service level classes are given priority

utilisation of the mail store processors. When an overload condition is detected by the overload controller then the overload controller configures the access controller to deny access to the platform to certain signals (for example, requests to open a low priority mailbox in the mail store). Different types of signals are assigned different priorities. Three alternative schemes are described below by way of example.

Scheme 1

In this scheme, different service classes are applied to different service providers. During overload conditions, as monitored by a load controller, those subscribing to a higher priority class of service level agreement are given priority over those requests coming from a Service Provider with a lower priority class. Hence, actions pertaining to the particular mailboxes used by customers of the higher priority class of service would receive preferential treatment. It is envisaged that a range of service classes will be defined, which will correspond to the overload mechanism achieving a specified quality of service for those classes. Examples of service classes are given in Table 1 below.

Service Class	***Description***
III	Guaranteed end user access 99.99% of the time Guaranteed Service Provider notifications within 30 seconds, 99.99% of the time
II	Guaranteed end user access 98.0 % of the time Guaranteed Service Provider notifications within 30 seconds, 90.0% of the time Guaranteed Service Provider notifications within 20 minutes, 99.99% of the time
I	No guaranteed end user access No guaranteed Service Provider notifications

Scheme 2

As in scheme 1 different service classes are used. In this case the different service classes are applied to individual mailboxes for particular subscribers. Mailbox subscribers subscribing to higher service level classes are given priority

over requests for action associated with individual mailbox users with a lower priority class.

Scheme 3

In this scheme, different priorities are applied to certain types of transaction across the service provider interface. For example, the ability to deposit a message from a caller into a mailbox could take priority over a change of profile request from the user depending upon the load conditions prevailing.

As a further alternative, elements of two or more of these schemes may be combined, so that the priority given to a certain signal at the service provider interface may depend upon a weighted combination of the service provider class, the subscriber class and the signalling type priority.

A service provider application which itself becomes overloaded may also request that the mail system imposes an overload control scheme (such as those described above) to protect that application from overload.

In operation, which ever scheme is used, the overload controller monitors an appropriate one or more loading parameters. In the present example this parameter is the rate of transactions between the service provider applications and the mailboxes which pass through the access controller. This mechanism provides a measure of mail system processing usage. Other examples of measuring processing load may be based on directly measuring the load on the platform Central Processor Units and other hardware devices (this information is usually available from operating systems). Alternatively, other load measures, such as the amount of storage utilised (disk and memory) can be measured. Again this information is provided by most operating systems (e.g. Windows 95 and UNIX both provide system information). Threshold levels for system loading are defined for the mail system. The thresholds are defined such that a defined quality of service can be met for particular service classes (seeTable 1), and are dependent upon the platform and Mail system being used, and the customer/service mix for the particular implementation. As these threshold levels are reached, the overload control management imposes overload control. The following controls can be imposed:

- refusing new requests by Service Providers to open low priority mailboxes,
- closing Service Provider access to low priority open mailboxes,

over requests for action associated with individual mailbox users with a lower priority class.

Scheme 3

In this scheme, different priorities are applied to certain types of transaction across the service provider interface. For example, the ability to deposit a message from a caller into a mailbox could take priority over a change of profile request from the user depending upon the load conditions prevailing.

As a further alternative, elements of two or more of these schemes may be combined, so that the priority given to a certain signal at the service provider interface may depend upon a weighted combination of the service provider class, the subscriber class and the signalling type priority.

A service provider application which itself becomes overloaded may also request that the mail system imposes an overload control scheme (such as those described above) to protect that application from overload.

In operation, which ever scheme is used, the overload controller monitors an appropriate one or more loading parameters. In the present example this parameter is the rate of transactions between the service provider applications and the mailboxes which pass through the access controller. This mechanism provides a measure of mail system processing usage. Other examples of measuring processing load may be based on directly measuring the load on the platform Central Processor Units and other hardware devices (this information is usually available from operating systems). Alternatively, other load measures, such as the amount of storage utilised (disk and memory) can be measured. Again this information is provided by most operating systems (e.g. Windows 95 and UNIX both provide system information). Threshold levels for system loading are defined for the mail system. The thresholds are defined such that a defined quality of service can be met for particular service classes (seeTable 1), and are dependent upon the platform and Mail system being used, and the customer/service mix for the particular implementation. As these threshold levels are reached, the overload control management imposes overload control. The following controls can be imposed:

- refusing new requests by Service Providers to open low priority mailboxes,
- closing Service Provider access to low priority open mailboxes,

- refusing end user access to low priority mailboxes,
- queuing control messages and data transfers on the next highest priority open mailboxes.

If the next load threshold level is reached, the same overload controls will be imposed on the next highest priority mailboxes. If instead, the system load reduces, the overload controls set at that level are removed in reverse order to that in which they were applied.

In addition to responding to overload conditions within the platform, the overload controller may also be programmed to recognise and respond to signals from a service platform indicating that that platform is overloaded. The overload controller might then, for example, inhibit the transmission of notifications from the messaging platform to the overloaded service platform.

Figures 3a to 3y are SDL (Specification and Description Language [ITU-T Z.100]) diagrams showing in further detail the implementation of the messaging platform, including the overload controller. The structures shown may be compiled using commercially available software tools to provide code for running on the processors of the messaging platform. A description of the SDL diagrams follows:

Figure 3a: Diagram Structure

Describes the structure of the following figures 3b to 3y.

Figures 3b-3d: System mail_system 1-3

Defines the static limits of the described mail system, and type definitions and operators for variables used.

Figure 3e: System mail_system 4

Defines all of the control signals used in the mail system.

Figure 3f: System mail_system 5

Defines the two main blocks of the system: the mail application block (i.e. the service provider platform) and the mail system block (i.e. the messaging platform). The interfaces between the blocks (the mail system API), and to/from the external environment are also defined.

- refusing end user access to low priority mailboxes,
- queuing control messages and data transfers on the next highest priority open mailboxes.

If the next load threshold level is reached, the same overload controls will be imposed on the next highest priority mailboxes. If instead, the system load reduces, the overload controls set at that level are removed in reverse order to that in which they were applied.

In addition to responding to overload conditions within the platform, the overload controller may also be programmed to recognise and respond to signals from a service platform indicating that that platform is overloaded. The overload controller might then, for example, inhibit the transmission of notifications from the messaging platform to the overloaded service platform.

Figures 3a to 3y are SDL (Specification and Description Language [ITU-T Z.100]) diagrams showing in further detail the implementation of the messaging platform, including the overload controller. The structures shown may be compiled using commercially available software tools to provide code for running on the processors of the messaging platform. A description of the SDL diagrams follows:

Figure 3a: Diagram Structure

Describes the structure of the following figures 3b to 3y.

Figures 3b-3d: System mail_system 1-3

Defines the static limits of the described mail system, and type definitions and operators for variables used.

Figure 3e: System mail_system 4

Defines all of the control signals used in the mail system.

Figure 3f: System mail_system 5

Defines the two main blocks of the system: the mail application block (i.e. the service provider platform) and the mail system block (i.e. the messaging platform). The interfaces between the blocks (the mail system API), and to/from the external environment are also defined.

Figure 3g: Block mail_application_block 1

Defines the processes and interfaces within the mail application block. In this case, only the mail application processes are defined.

Figure 3h-3j: Process mail_application 1-3

Defines all of the control signals, variables, states, states transitions and actions of the mail application necessary to illustrate the present invention. The process contains one state ('alive'), and acts on a number of events as follows:

user_mailbox_open: A request from a user (or some other stimulus) to open a mailbox results in a request to the access controller to open a mailbox.

user_mailbox_close: A request from a user (or some other stimulus) to close a mailbox results in a request to the access controller to close a mailbox.

user_mailbox_request: A request from a user (or some other stimulus) to perform an action on an open mailbox usually results in a request to the access controller to perform an action on an open mailbox. If however, the mail system is overloaded, then the request will be queued.

mail_application_overload_detected: An indication from the service provider platform (or some other stimulus) that the mail application is overloaded results in a request to the access controller to impose overload controls on open mailboxes opened by that application that are below a specified service class. Requests to these mailboxes are not queued.

mail_application_overload_ceased: An indication from the service provider platform (or some other stimulus) that the mail application overload has ceased results in a request to the access controller to remove overload controls on open mailboxes opened by that application that are above a specified service class.

UM_opened: Confirms that a mailbox has been opened.

UM_closed: Confirms that a mailbox has been closed.

UM_open_denied: Indicates that a request to open a mailbox has been denied – for example because overload controls have been imposed.

UM_notification: Indicates a notification from a mailbox, which is passed back to the application user (if appropriate). The notification could be the result of an earlier mail application request on the mailbox, or another (e.g. timer) event.

Figure 3g: Block mail_application_block 1

Defines the processes and interfaces within the mail application block. In this case, only the mail application processes are defined.

Figure 3h-3j: Process mail_application 1-3

Defines all of the control signals, variables, states, states transitions and actions of the mail application necessary to illustrate the present invention. The process contains one state ('alive'), and acts on a number of events as follows:

user_mailbox_open: A request from a user (or some other stimulus) to open a mailbox results in a request to the access controller to open a mailbox.

user_mailbox_close: A request from a user (or some other stimulus) to close a mailbox results in a request to the access controller to close a mailbox.

user_mailbox_request: A request from a user (or some other stimulus) to perform an action on an open mailbox usually results in a request to the access controller to perform an action on an open mailbox. If however, the mail system is overloaded, then the request will be queued.

mail_application_overload_detected: An indication from the service provider platform (or some other stimulus) that the mail application is overloaded results in a request to the access controller to impose overload controls on open mailboxes opened by that application that are below a specified service class. Requests to these mailboxes are not queued.

mail_application_overload_ceased: An indication from the service provider platform (or some other stimulus) that the mail application overload has ceased results in a request to the access controller to remove overload controls on open mailboxes opened by that application that are above a specified service class.

UM_opened: Confirms that a mailbox has been opened.

UM_closed: Confirms that a mailbox has been closed.

UM_open_denied: Indicates that a request to open a mailbox has been denied – for example because overload controls have been imposed.

UM_notification: Indicates a notification from a mailbox, which is passed back to the application user (if appropriate). The notification could be the result of an earlier mail application request on the mailbox, or another (e.g. timer) event.

queue_UM_requests: Indicates that the access controller has imposed overload controls on a mailbox, and that the mail application should not make further requests on that mailbox. The mail application will queue subsequent requests.

resume_UM_requests: Indicates that the access controller has removed overload controls on a mailbox, and that the mail application can make further requests on that mailbox. Any pending requests in the queue will be immediately sent to the mail system.

Figure 3k: Block mail_system_block 1

Defines the processes and interfaces within the mail system block. In this case, there are three processes: the single access controller, the single overload controller, and the multiple user mailboxes.

Figure 3l-3p: Process access controller 1-5

Defines all of the control signals, variables, procedures, states, states transitions and actions of the access controller necessary to illustrate the present invention. The process contains one state ('alive') that is entered after the 'tr_timer' has been set. The process acts on a number of events as follows:

UM_open: A request from a mail application to open a mailbox is received. The transaction counter is incremented. If the mailbox has a higher priority ('class') than that set by the overload control mechanism, the mailbox is opened (by creating a user mailbox process), the identity of the mailbox is added to the linked list of open mailboxes, and the actions are confirmed to the mail application. Otherwise, the mail application is informed that the request is denied.

UM_close: A request from a mail application to close a mailbox is received. The transaction counter is incremented. The mailbox is closed - by terminating the user mailbox process, the identity of the mailbox is removed from the linked list of open mailboxes, and the actions are confirmed to the mail application.

UM_request: A request from a mail application to perform an operation on an open mailbox is received. The transaction counter is incremented. If the overload control is not applied to that mailbox, then the request is passed onto the mailbox. Otherwise, the request is placed in an request queue.

set_UM_class_bar: A request from the overload controller to set the priority level ('class') at which mailboxes may be opened or controlled is received. If the priority

level is higher than the current priority level (the 'current_class_bar'), then mail applications with open mailboxes below the priority level are requested to queue further requests to that mailbox (procedure 'queue_transactions'), the current priority level is raised to the new level. If the priority level is lower than the current priority level (the 'current_class_bar'), then any queued requests received from mail applications are unqueued (procedure 'unqueue_requests'). Following this, mail applications with open mailboxes above the priority level are requested to resume further requests to that mailbox (procedure 'resume_transactions'), and the current priority level is lowered to the new level.

notification: A notification from an open mailbox is received. The transaction counter is incremented, and the notification is passed on to the relevant mail application.

impose_UM_load_control: A request from a mail application to impose load control on open mailboxes that were opened by the application is received. Mailboxes with a class less than or equal to the specified level are requested to queue up notifications, if not already doing so. Mailboxes with a class greater than the specified level are requested to resume notifications, if not already asked to do so. This mechanism does not override any controls imposed by the overload controller.

tr_timer: A transaction timer expiry event is received. The transaction timer is set to expire every second. The current transaction rate is thus the value of the transaction counter, per second. This overload controller is informed of the current transaction rate, the transaction counter is reset to zero and the tr_timer reset.

Figure 3q: Procedure get_UM_class 1

Defines the procedure for retrieving the provisioned service class of a mailbox from the management system for the mail system.

Figure 3r: Procedure queue_transactions 1

Defines the procedure for informing the mail application to queue requests to open mailboxes, and informing the mailbox to queue notifications, when overload controls are imposed.

Figure 3s: Procedure resume_transactions 1

Defines the procedure for informing the mail application to queue requests to open mailboxes, and informing the mailbox to queue notifications, when overload controls are imposed.

Figure 3t: Procedure unqueue_requests 1

Defines the procedure for unqueueing requests for operations on mailboxes when overload controls have been removed.

Figure 3u: Process overload_controller 1

Defines all of the control signals, variables, procedures, states, states transitions and actions of the overload controller necessary to illustrate the present invention. The process contains one state ('monitoring') that is entered after the threshold levels for the overload controls have been set in the threshold levels table. The process acts on a number of events as follows:

transaction_rate: An indication of the current transaction rate at the access controller is received. If this rate has fallen below the current minimum level, then the current overload priority ('UM_class_bar') is decremented (if it is not already at the lowest level). The threshold levels are then adjusted accordingly - from the threshold table, and the access control is requested to reduce or remove the overload control. If, alternatively, the rate has risen above the current maximum level, then the current overload priority ('UM_class_bar') is incremented (if it is not already at the highest level). The threshold levels are then adjusted accordingly and the access control is requested to increase the level of overload control.

Figure 3v: Procedure set_thresholds 1

Defines the procedure for setting the threshold levels in the overload control threshold table. These levels define the points at which the oveload controls will be imposed or removed.

Figure 3w-3x: Process user_mailbox 1-2

Defines all of the control signals, variables, procedures, states, states transitions and actions of the user mailbox necessary to illustrate the present invention. The process contains one state ('alive'), and acts on a number of events as follows:

queue_notifications: An request to queue notifications is received from the access controller. A flag is set indicate that future notifications should be queued.

resume_notifications: An request to resume notifications is received from the access controller. A flag is cleared indicate that future notifications should no longer be queued, and any queued notifications are unqueued.

request: An request to perform an operation is received from the mail application via the access controller. The operation is performed as requested (procedure 'perform_request').

event: An indication that an event has occurred in the mailbox (e.g. the result of an operation or a timer expiry) is received. If overload control has been imposed, then the notification of the event to the mail application is queued. Otherwise, the notification is sent to the mail application via the access controller.

stop_mailbox: An request to stop the user mailbox process is received from the access controller. The user mailbox process terminates, closing the mailbox.

Figure 3y: Procedure perform_request 1

Defines the procedure for the mailbox to perform an operation requested by the mail application.

Figure 4 illustrates an example of hardware systems used to implement the present invention. It will be understood that the components and architecture shown here are given by way of example only, and many alternative components and architectures may be used equivalently. For example, rather than using platforms built from rack-mounted processors and storage devices, some or all of the platforms might be implemented on computer workstations. Such workstations may be joined together by a network to provide a distributed computing system. As a further example of the alternatives available, storage devices are by no means limited to conventional hard disks, but might alternatively or in addition use magneto-optical or optical storage media. Also, as previously stated, one or more of the "service platforms" may be embodied in a customer

terminal that exercises a control function, for example in a personal computer, or in a so-called internet appliance such as a television set-top box arranged to provide Internet access, or in an intelligent telephone that provides the necessary control capabilities.

In Figure 4, end users connect to the Service Provider platform by telephone or computer. One or more Service Provider platforms are connected to the Mail Store platform. The Service Provider platform runs the mail applications (for example to collect messages, and permit retrieval of messages), interfaces to the end users, and sends control and data signals to the Mail Store platform to store and retrieve messages held in electronic format.

It is possible that the Service Provider platform and Mail Store Platform reside on the same computing node. In this case the buses B1 and B2 can be joined directly, therefore removing the need for interface I3, the Mail Store interface shelf (S3) and the Service Provider interface shelf (M3). Alternatively the platforms may be connected to each other via a network, for example the Internet, an ATM network, or a local area network LAN.

A description of the system follows:

Interfaces

Interface I1: This is a standard telephony interface, which allows end users to record or retrieve voice messages. The interface will usually be provided via a telephony network.

Interface I2: This is an electronic data interface to a computer (e.g. TCP/IP), which allows end users to record or retrieve voice or electronic messages. The interface will usually be provided via a data network such as the internet.

Interface I3: This is an electronic data interface which transports the control and data signals between the Service Provider and Mail Store platforms. This is typically implemented as an ethernet, frame-relay or X.25 data link, running the TCP/IP protocol. Control and Data messages may be carried using suitable protocols, such as the Internet File Transfer Protocol (FTP).

Interface I4: This is an electronic data interface to the Service Provider platform management system (for example, a customer management system or simply a computer terminal). The protocol used with this interface will typically be the internet Simple Network Management Protocol (SNMP) running over TCP/IP.

Interface I5: This is an electronic data interface to the Mail Store platform management system (for example, a network management system or simply a computer terminal). The protocol used with this interface will typically be the internet Simple Network Management Protocol (SNMP) running over TCP/IP.

Interface I6: This is an electronic data interface which transports the control and data signals between the Mail Store platforms. This allows the Mail System to be distributed across a number of sites (e.g. for resilience and performance reasons). This is typically implemented as a frame-relay or X.25 data link, running the TCP/IP protocol. Control and Data messages may be carried using suitable protocols, such as the Internet File Transfer Protocol (FTP).

Service Provider Platform Components

The Service Provider platform comprises a number of shelves, which are specialised to meet particular needs, and a bus (bus B1) that connects the shelves. A shelf is either a processor shelf (consisting of processor, memory, clocks, I/O devices, etc.) or a storage shelf (consisting of hard disk storage, disk controllers, etc.).

Shelf S1: Mail Applications Processor shelf.
This runs the various applications that interface the end users and other applications to the mail system.

Shelf S2: Platform Management Processor shelf.
This runs the necessary management applications for the service provider platform, and interfaces to an external management system via I4.

Shelf S3: Mail System Interface Processor shelf.

This provides the physical termination of the interface (I3) to the Mail System platform, and runs the software to interface the protocols to the rest of the Service Provider platform.

Shelf S4: Interactive Voice Response Processor shelf.
This runs the software that interacts directly with end uses, usually over the telephony interface (I1) via the Telephony Interface processor shelf (although it can be used to interact with users over the internet interface). This shelf is able to play announcements to end users, and record voice messages into electronic format (e.g. PC '.wav' format), that will ultimately be stored in the Mail System.

Shelf S5: Telephony Processor shelf.
This provides the physical termination of the interface (I1) to the telephony network, and runs the software to interface the protocols and bearer circuits to the rest of the Service Provider platform.

Shelf S6: Internet Processor shelf.
This provides the physical termination of the interface (I2) to the internet network, and runs the software to interface the control and data protocols to the rest of the Service Provider platform.

Shelf S7: Runtime Data Storage shelf.
Provides hard disk storage for the various processor shelves to use at run time.

Shelf S8: Audit Data Storage shelf.
Provides resilient hard disk storage for system data that needs to be held for logging and auditing purposes. This is usually stored by the processor shelves, and retrieved by the external management system via the Platform Management processor shelf (S2).

Mail Store Platform Components

The Mail Store platform comprises a number of shelves, which are specialised to meet particular needs, and a bus (bus B2) that connects the shelves. A shelf is

either a processor shelf (consisting of processor, memory, clocks, I/O devices, etc) or a storage shelf (consisting of hard (disk) storage, disk controllers, etc).

Shelf M1: Mailbox Controllers Processor shelf.
This runs the software used to store and retrieve messages and other information in user mailboxes.

Shelf M2: Platform Management Processor shelf.
This runs the necessary management applications for the service provider platform, and interfaces to an external management system via I5.

Shelf M3: Service Provider Interface Processor shelf.
This provides the physical termination of the interface (I3) from one or more Service Provider platforms, and runs the software to interface the protocols to the rest of the Mail System platform.

Shelf M4 Access and Overload Controller Processor shelf.
This runs the access controller and overload controller process software for the Mail System.

Shelf M5: Mail System Interface Processor shelf.
This provides the physical termination of the interface (I6) from one or more other Mail System platforms, and runs the software to interface the protocols to the rest of the Mail System platform.

Shelf M7: Mailbox Data Storage shelf.
Provides resilient hard disk storage for the user mailboxes. This is shelf is usually only accessed by the Mailbox Controllers shelf (M1) and the Platform Management processor shelf (M2).

Shelf M7: Runtime Data Storage shelf.
Provides hard disk storage for the various processor shelves to use at run time.

Shelf M8: Audit Data Storage shelf.

Provides resilient hard disk storage for system data that needs to be held for logging and auditing purposes. This is usually stored by the processor shelves, and retrieved by the external management system via the Platform Management processor shelf (M2).

5

CLAIMS

1. A messaging platform including:

a) a message store arranged to receive message data and to store said message data for subsequent retrieval,

characterised by

b) an overload controller responsive to an overload condition in the platform and arranged, in response to the said overload condition, to limit loading of the platform.

2. A messaging platform according to claim 1, further comprising:

c) a control interface arranged to receive control requests instructing transactions on the messaging platform,

and in which the overload controller includes means for denying at least some of the control requests in response to the overload condition.

3. A platform according to claim 2, in which the overload controller is programmed with criteria for applying different classes of service to control requests received at the control interface and the overload controller is arranged, in response to an overload condition on the platform, selectively to deny control requests depending on a class of service assigned in accordance with the said criteria to the control request.

4. A platform according to claim 3, in which the criteria apply a class of service selected depending on the identity of a service provider originating the said control requests

5. A platform according to claim 3 or 4 in which the criteria apply a class of service selected depending on the identity of a subscriber mailbox to which the control request applies

6. A platform according to any one of claims 3 to 5, in which the criteria apply different service classes depending on the transaction requested by the control request.

7. A messaging system comprising:

a service platform running a messaging service application; and

a messaging platform according to any one of the proceeding claims connected to the service platform via a control interface, and arranged to receive control requests from the service platform via the control interface.

8. A messaging system according to claim 7, in which the service platform is remote from the messaging platform.

9. A communications network including a messaging platform according to any one of claims 1 to 6, or a messaging system according to claim 7 or 8.

10. A method of operating a messaging platform including:

a) storing message data on the messaging platform:

b) subsequently outputting message data from the platform, thereby allowing retrieval of a corresponding message;

characterised by

c) detecting an overload condition on the platform and in response to the overload condition limiting loading of the platform.

11. A method according to claim 10, further comprising

d) receiving at the message platform control requests instructing a transaction on the messaging platform,

and in which the step of limiting loading of the platform includes denying at least some of the control requests.

12. A method according to claim 11, including applying different classes of service to the control requests, and in response to the overload condition selectively denying some only of the control requests depending on the class of service applied to the control requests.

13. A method according to claim 12, including applying different classes of service to control requests depending on the identity of an originating service provider.

14. A method according to claim 12 or 13, including applying different classes of service to control requests depending on identities of customer mailboxes to which the control requests apply.

15. A method according to claim 12, 13 or 14, including applying different classes of service to control requests depending on the transaction requested by the control request

16. A method according to claim 15, in which the messaging platform includes a plurality of mailboxes containing message data, each mailbox being switchable between an open state, in which message data may be written to or read from the mailbox, and a closed state, and in which the step of limiting loading includes allowing requests for the closing of a mailbox and denying requests for the opening of a mailbox.

ABSTRACT

A messaging platform, for example a platform storing email or voicemail, is provided with an overload controller which detects an overload condition and then acts to limit the loading of the platform. The overload controller may be associated with a control interface used, for example, for communication between a service provider and the message platform, and may selectively deny some of the requests received over that interface when the overload condition is detected.

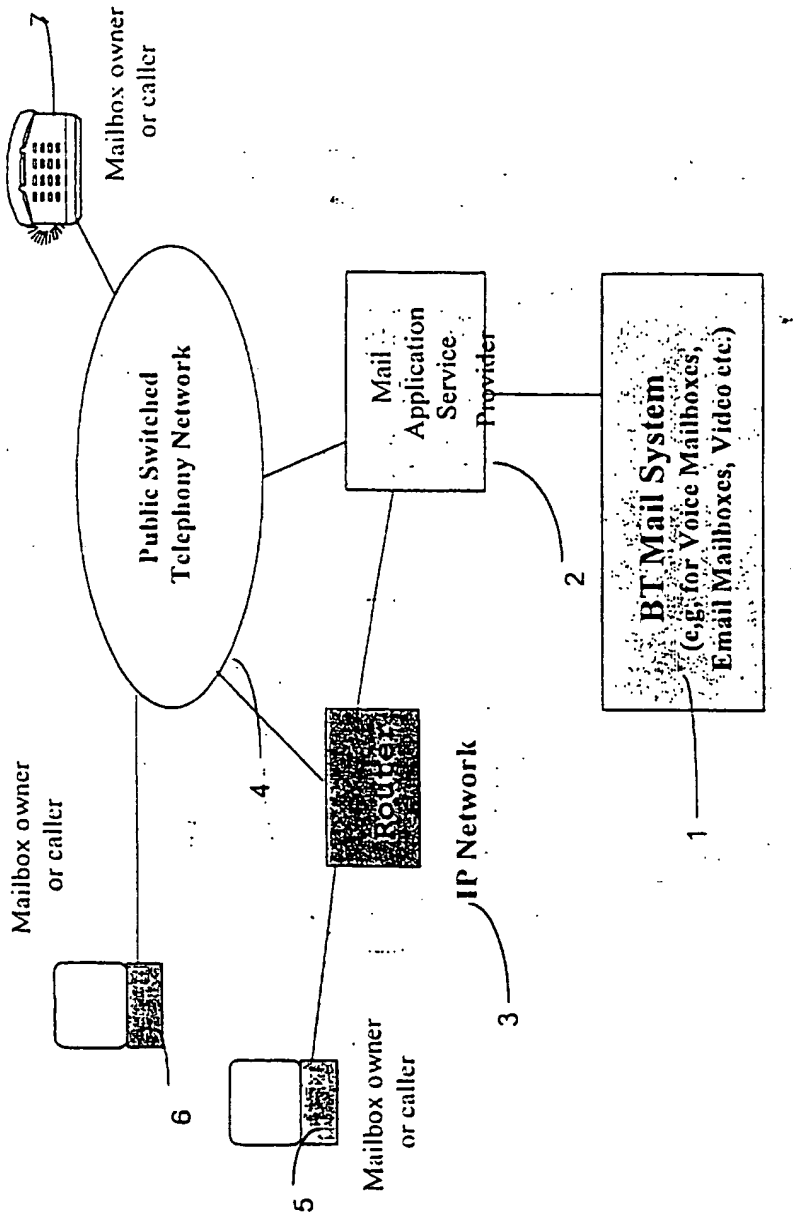

Mailbox owner or caller
7
Public Switched Telephony Network
Mail Application Service Provider
BT Mail System
(e.g. for Voice Mailboxes, Email Mailboxes, Video etc.)
2
1
4
Router
IP Network
3
Mailbox owner or caller
Mailbox owner or caller
6
5

Figure 1

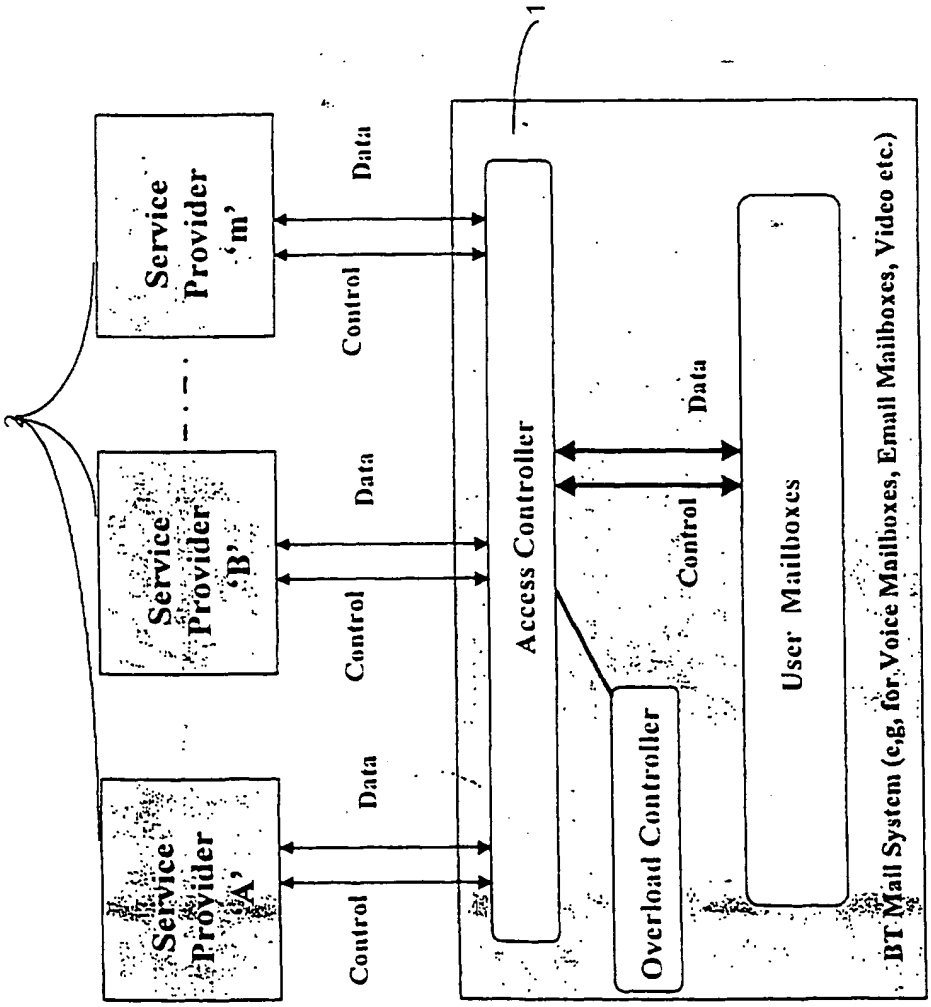
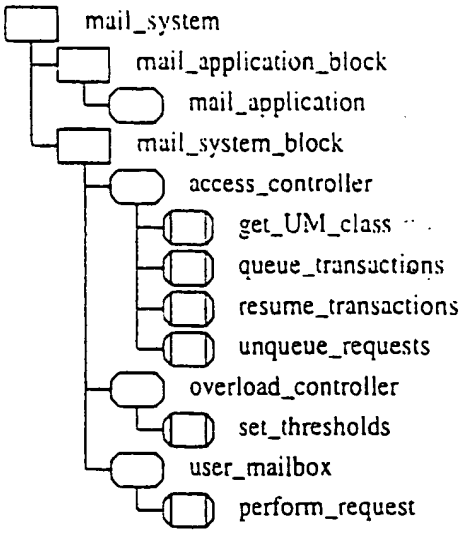

Service Provider 'A'
Service Provider 'B'
Service Provider 'm'
Control
Data
Control
Data
Control
Data
Access Controller
Overload Controller
Control
Data
User Mailboxes
BT Mail System (e.g. for Voice Mailboxes, Email Mailboxes, Video etc.)
1

Figure 2

Diagram Structure



Associated Documents

Figure 3a

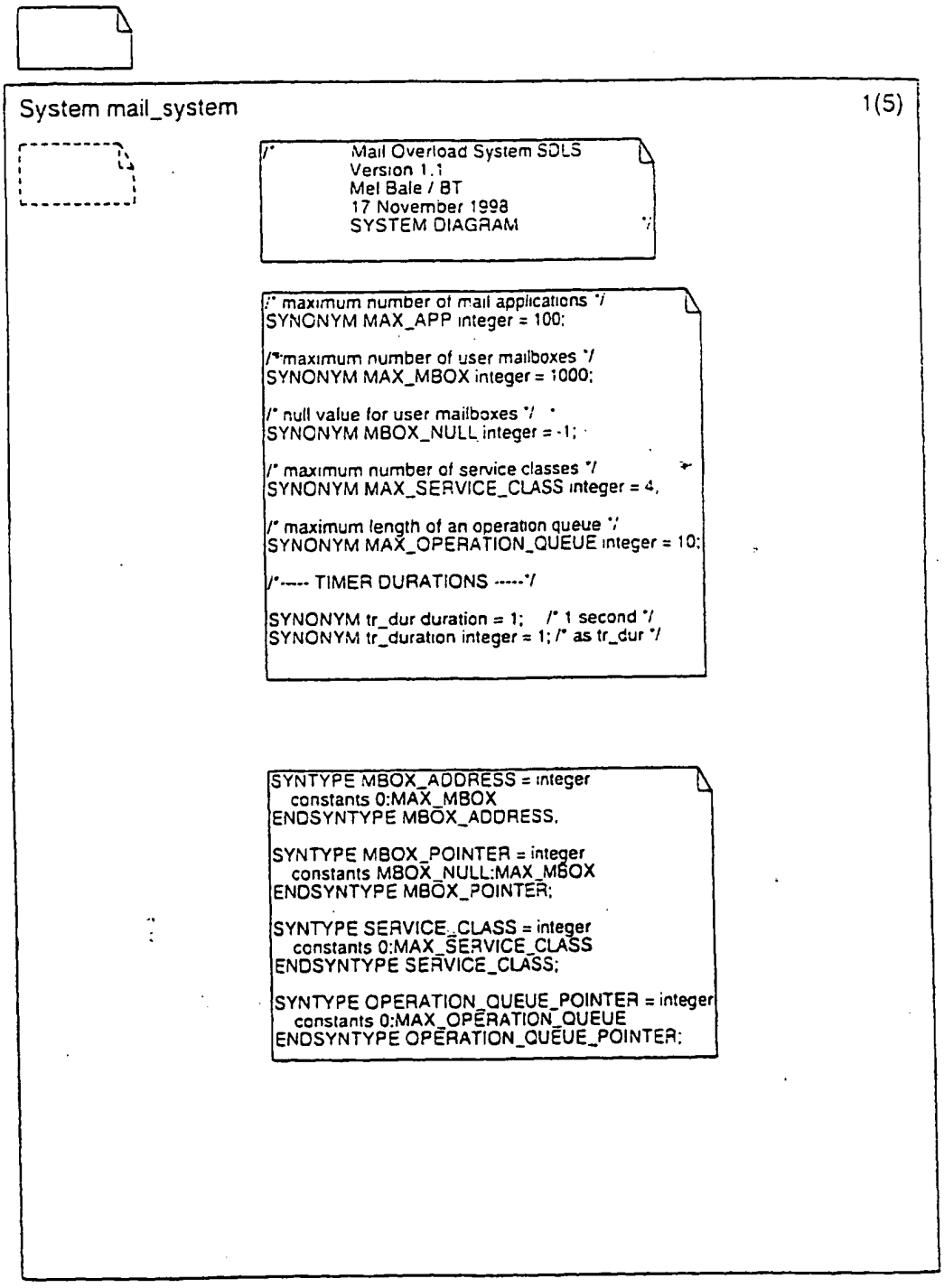

System mail_system
1(5)
/* Mail Overload System SDLS
Version 1.1
Mel Bale / BT
17 November 1998
SYSTEM DIAGRAM */
/* maximum number of mail applications */
SYNONYM MAX_APP integer = 100;
/*maximum number of user mailboxes */
SYNONYM MAX_MBOX integer = 1000;
/* null value for user mailboxes */
SYNONYM MBOX_NULL integer = -1;
/* maximum number of service classes */
SYNONYM MAX_SERVICE_CLASS integer = 4;
/* maximum length of an operation queue */
SYNONYM MAX_OPERATION_QUEUE integer = 10;
/*----- TIMER DURATIONS -----*/
SYNONYM tr_dur duration = 1; /* 1 second */
SYNONYM tr_duration integer = 1; /* as tr_dur */
SYNTYPE MBOX_ADDRESS = integer
constants 0:MAX_MBOX
ENDSYNTYPE MBOX_ADDRESS;
SYNTYPE MBOX_POINTER = integer
constants MBOX_NULL:MAX_MBOX
ENDSYNTYPE MBOX_POINTER;
SYNTYPE SERVICE_CLASS = integer
constants 0:MAX_SERVICE_CLASS
ENDSYNTYPE SERVICE_CLASS;
SYNTYPE OPERATION_QUEUE_POINTER = integer
constants 0:MAX_OPERATION_QUEUE
ENDSYNTYPE OPERATION_QUEUE_POINTER;

Figure 3b

System mail_system 2(5)

```
NEWTYPE CAUSE
  literals T_overload, T_unknown;
ENDNEWTYPE CAUSE;

NEWTYPE OPERATION_STRUCT
   struct
      owner pid;
      mbox_id MBOX_ADDRESS;
      class SERVICE_CLASS;
      info charstring;
ENDNEWTYPE OPERATION_STRUCT;

NEWTYPE OPERATION_QUEUE
   array(OPERATION_QUEUE_POINTER, OPERATION_STRUCT)
   adding
   operators
   /* return true if there are no more operations */
   is_last : OPERATION_QUEUE -> boolean;

   /* add an operation to the queue
      - return true if queue is full */
   add_op : OPERATION_QUEUE, OPERATION_STRUCT -> boolean;

   /* delete the current operation from the queue
      - return true if queue is empty */
   delete_op : OPERATION_QUEUE -> boolean;

   /* get the first operation structure in the list */
   get_first_op : OPERATION_QUEUE -> OPERATION_STRUCT;

   /* get the next operation structure in the list */
   get_next_op : OPERATION_QUEUE -> OPERATION_STRUCT;

ENDNEWTYPE OPERATION_QUEUE;
```

Figure 3c

System mail_system 3(5)

```
NEWTYPE MBOX_STRUCT
   struct
      class SERVICE_CLASS;
      mbox_id MBOX_ADDRESS,
      process_id pid;
      opener_id pid,
      opener_set_class SERVICE_CLASS,
      next_in_list MBOX_POINTER,
      queue boolean;
      op_queue OPERATION_QUEUE,
ENDNEWTYPE MBOX_STRUCT;

NEWTYPE MBOX_LIST
   array(MBOX_ADDRESS, MBOX_STRUCT)
 adding
   operators
   /* return true if there are no more mailboxes */
   is_last : MBOX_LIST -> boolean;

   /* add an open mailbox struct to the list
      - return true if list is full */
   add_mbox : MBOX_LIST, MBOX_STRUCT -> boolean,

   /* update an open mailbox struct to the list
      - return true if list is full */
   update_mbox : MBOX_LIST, MBOX_STRUCT -> boolean;

   /* delete an open mailbox struct from the list */
   delete_mbox : MBOX_LIST, MBOX_STRUCT -> boolean;

   /* get the first open mailbox structure in the list */
   get_first_mbox : MBOX_LIST -> MBOX_STRUCT;

   /* get the next open mailbox structure in the list */
   get_next_mbox : MBOX_LIST -> MBOX_STRUCT;

   /* get a specific open mailbox structure from the list */
   get_mbox : MBOX_LIST, MBOX_ADDRESS -> MBOX_STRUCT,
ENDNEWTYPE MBOX_LIST;

NEWTYPE RATE_STRUCT
   struct
      min_rate integer;
      max_rate integer;
ENDNEWTYPE RATE_STRUCT,

NEWTYPE RATE_ARRAY
   array(SERVICE_CLASS, RATE_STRUCT)
ENDNEWTYPE RATE_ARRAY;
```

Figure 3d

System mail_system 4(5)

```
SIGNAL  /*----- SIGNALS for routes 1 -----*/
user_mailbox_open(MBOX_ADDRESS),
user_mailbox_close(MBOX_ADDRESS),
user_mailbox_opened(MBOX_ADDRESS),
user_mailbox_closed(MBOX_ADDRESS),
user_mailbox_request(MBOX_ADDRESS,charstring),
user_mailbox_notification(MBOX_ADDRESS,charstring),
user_mailbox_open_denied(MBOX_ADDRESS,CAUSE),
mail_application_overload_detected,
mail_application_overload_ceased;


SIGNAL  /*----- SIGNALS for route 2 -----*/
UM_open(pid, MBOX_ADDRESS),
UM_close(pid, MBOX_ADDRESS),
UM_opened(MBOX_ADDRESS),
UM_open_denied(MBOX_ADDRESS,CAUSE),
UM_closed(MBOX_ADDRESS),
UM_request(OPERATION_STRUCT),
UM_notification(OPERATION_STRUCT),
impose_UM_load_control(pid,SERVICE_CLASS),
queue_UM_requests(MBOX_ADDRESS),
resume_UM_requests(MBOX_ADDRESS);

SIGNAL  /*----- SIGNALS for route 3 -----*/
queue_notifications,
resume_notifications,
stop_mailbox,
request(OPERATION_STRUCT),
notification(OPERATION_STRUCT);

SIGNAL  /*----- SIGNALS for route 4 -----*/
transaction_rate(integer),
set_UM_class_bar(SERVICE_CLASS);

SIGNAL  /*----- SIGNALS for route 5 -----*/
UM_service_class_query(MBOX_ADDRESS),
UM_service_class_result(MBOX_ADDRESS,SERVICE_CLASS);

SIGNAL  /*----- SIGNALS internal to User_Mailbox -----*/
event(OPERATION_STRUCT);
```

```
SIGNALLIST Slist_1_out =
user_mailbox_opened,
user_mailbox_open_denied,
user_mailbox_closed
user_mailbox_notification,

SIGNALLIST Slist_1_in =
user_mailbox_open,
user_mailbox_close,
user_mailbox_request,
mail_application_overload_detected,
mail_application_overload_ceased;

SIGNALLIST Slist_2_out =
UM_opened,
UM_open_denied,
UM_closed,
UM_notification,
queue_UM_requests,
resume_UM_requests;

SIGNALLIST Slist_2_in =
UM_open,
UM_close,
UM_request,
impose_UM_load_control;

SIGNALLIST Slist_3_out =
notification;

SIGNALLIST Slist_3_in =
queue_notifications,
resume_notifications,
request,
stop_mailbox;

SIGNALLIST Slist_4_out =
set_UM_class_bar;

SIGNALLIST Slist_4_in =
transaction_rate;

SIGNALLIST Slist_5_out =
UM_service_class_query;

SIGNALLIST Slist_5_in =
UM_service_class_result;
```

Figure 3e

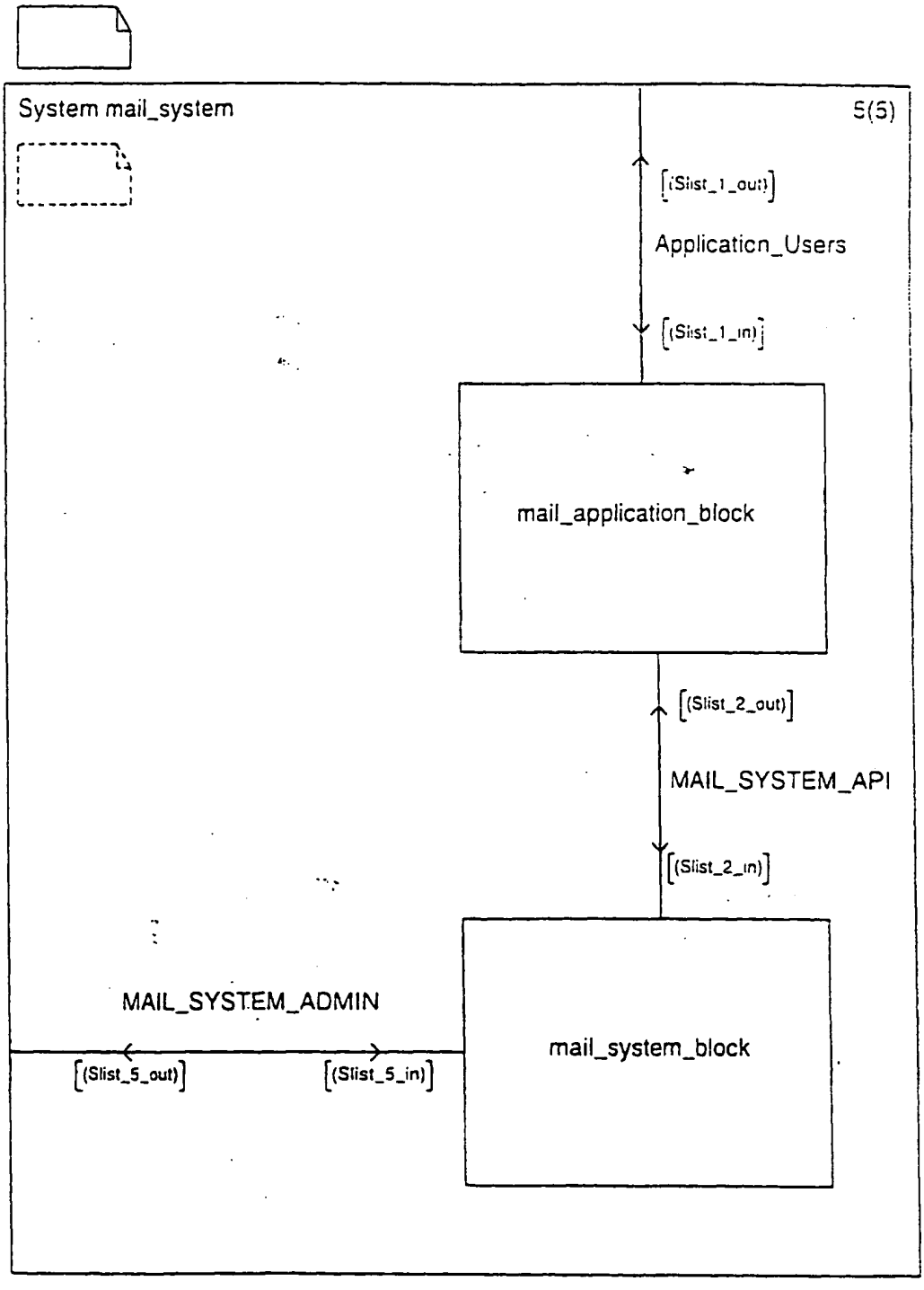

System mail_system
5(5)
(Slist_1_out)
Application_Users
(Slist_1_in)
mail_application_block
(Slist_2_out)
MAIL_SYSTEM_API
(Slist_2_in)
MAIL_SYSTEM_ADMIN
mail_system_block
(Slist_5_out)
(Slist_5_in)

Figure 3f

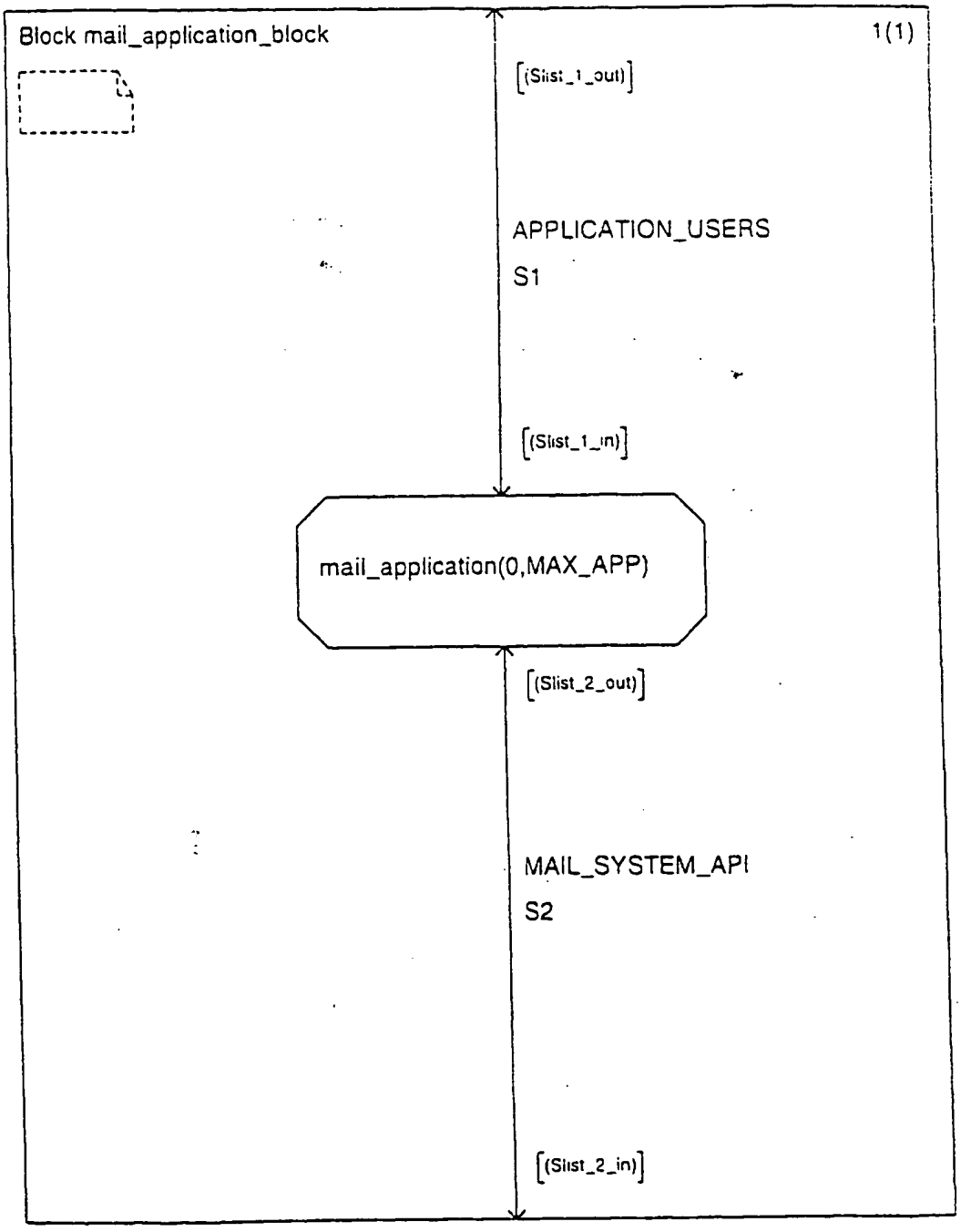

Block mail_application_block
1(1)
(Slist_1_out)
APPLICATION_USERS
S1
(Slist_1_in)
mail_application(0,MAX_APP)
(Slist_2_out)
MAIL_SYSTEM_API
S2
(Slist_2_in)

Figure 3g

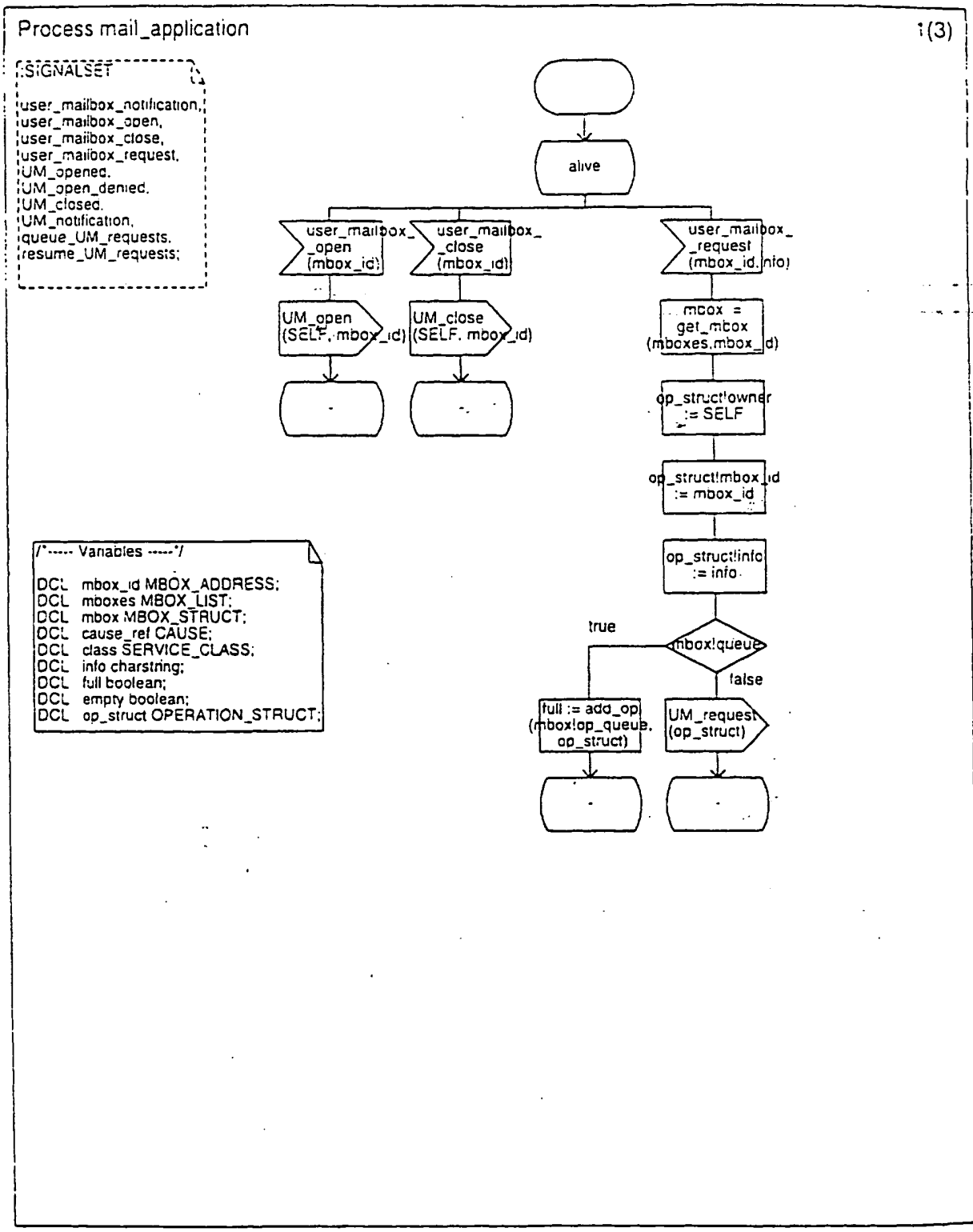

Process mail_application
1 (3)
SIGNALSET
user_mailbox_notification,
user_mailbox_open,
user_mailbox_close,
user_mailbox_request,
UM_opened,
UM_open_denied,
UM_closed,
UM_notification,
queue_UM_requests,
resume_UM_requests;
alive
user_mailbox_ _open (mbox_id);
user_mailbox_ _close (mbox_id);
user_mailbox_ _request (mbox_id, info)
UM_open (SELF, mbox_id)
UM_close (SELF, mbox_id)
mbox = get_mbox (mboxes,mbox_id)
op_struct!owner := SELF
op_struct!mbox_id := mbox_id
op_struct!info := info
/*----- Variables -----*/
DCL mbox_id MBOX_ADDRESS;
DCL mboxes MBOX_LIST;
DCL mbox MBOX_STRUCT;
DCL cause_ref CAUSE;
DCL class SERVICE_CLASS;
DCL info charstring;
DCL full boolean;
DCL empty boolean;
DCL op_struct OPERATION_STRUCT;
true
mbox!queue
false
full := add_op (mbox!op_queue, op_struct)
UM_request (op_struct)

Figure 3h

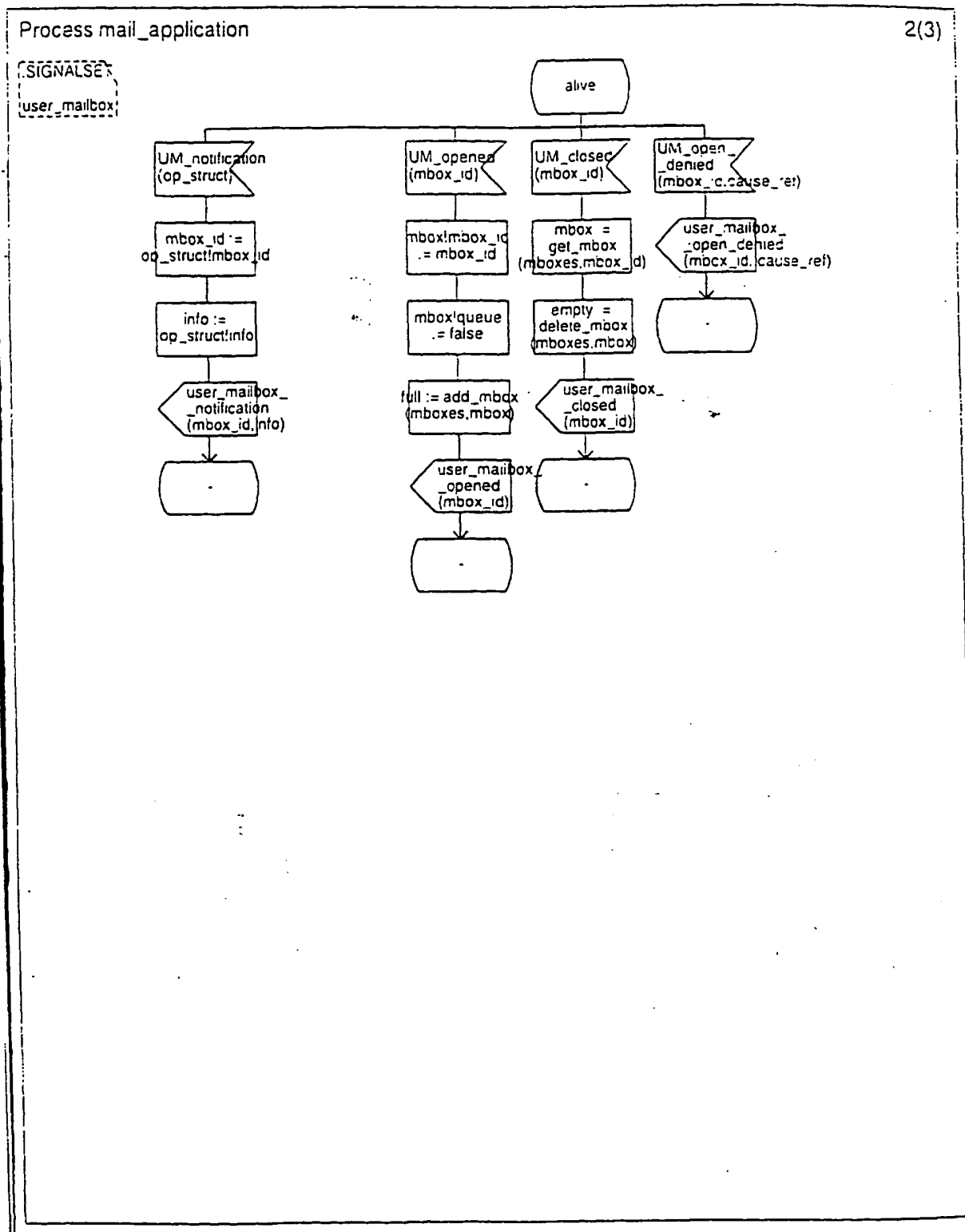

Process mail_application
2(3)
SIGNALSET
user_mailbox
alive
UM_notification (op_struct)
UM_opened (mbox_id)
UM_closed (mbox_id)
UM_open_ _denied (mbox_id,cause_ref)
mbox_id := op_struct!mbox_id
mbox!mbox_id := mbox_id
mbox = get_mbox (mboxes,mbox_id)
user_mailbox_ _open_denied (mbox_id, cause_ref)
info := op_struct!info
mbox!queue := false
empty = delete_mbox (mboxes,mbox)
user_mailbox_ _notification (mbox_id, info)
full := add_mbox (mboxes,mbox)
user_mailbox_ _closed (mbox_id)
user_mailbox_ _opened (mbox_id)

Figure 3i

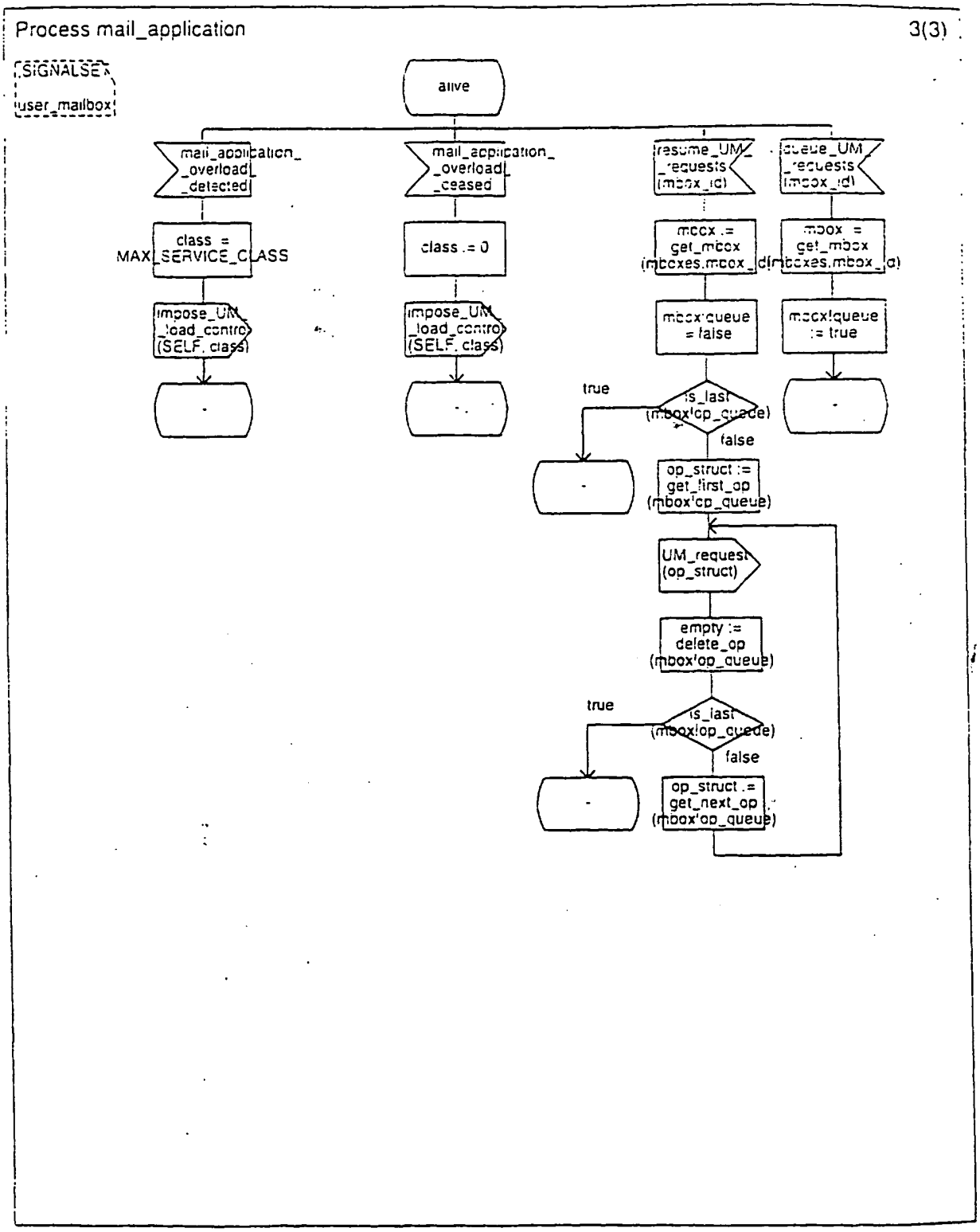

Process mail_application
3(3)
SIGNALSET
user_mailbox
alive
mail_application_overload_detected
mail_application_overload_ceased
resume_UM_requests (mbox_id)
queue_UM_requests (mbox_id)
class := MAX_SERVICE_CLASS
class := 0
mbox := get_mbox (mboxes.mbox_id)
mbox := get_mbox (mboxes.mbox_id)
impose_UM_load_control (SELF, class)
impose_UM_load_control (SELF, class)
mbox!queue := false
mbox!queue := true
true
is_last (mbox!op_queue)
false
op_struct := get_first_op (mbox!op_queue)
UM_request (op_struct)
empty := delete_op (mbox!op_queue)
true
is_last (mbox!op_queue)
false
op_struct := get_next_op (mbox!op_queue)

Figure 3j

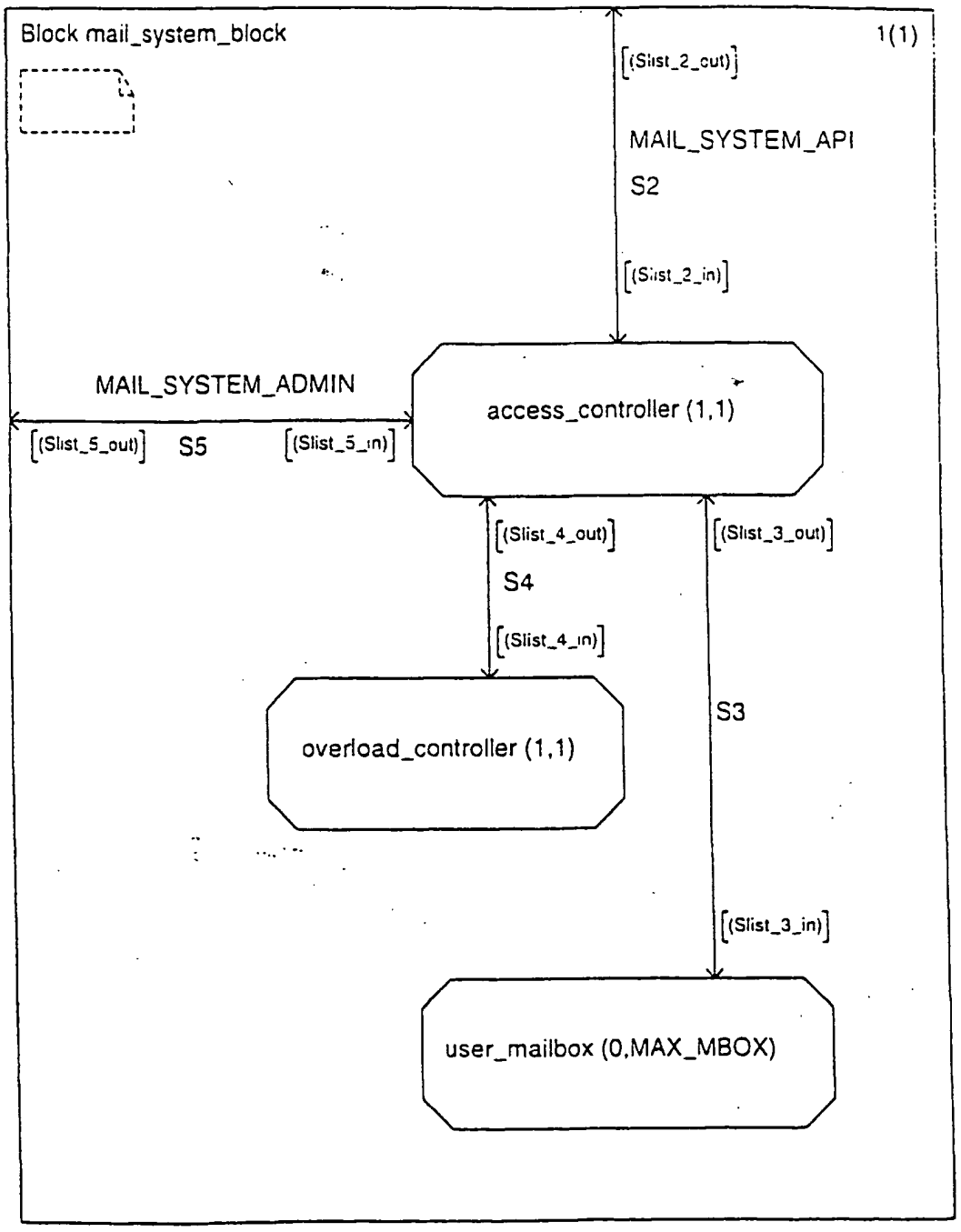

Block mail_system_block
1(1)
[(Slist_2_out)]
MAIL_SYSTEM_API
S2
[(Slist_2_in)]
MAIL_SYSTEM_ADMIN
access_controller (1,1)
[(Slist_5_out)]
S5
[(Slist_5_in)]
[(Slist_4_out)]
[(Slist_3_out)]
S4
[(Slist_4_in)]
S3
overload_controller (1,1)
[(Slist_3_in)]
user_mailbox (0,MAX_MBOX)

Figure 3k

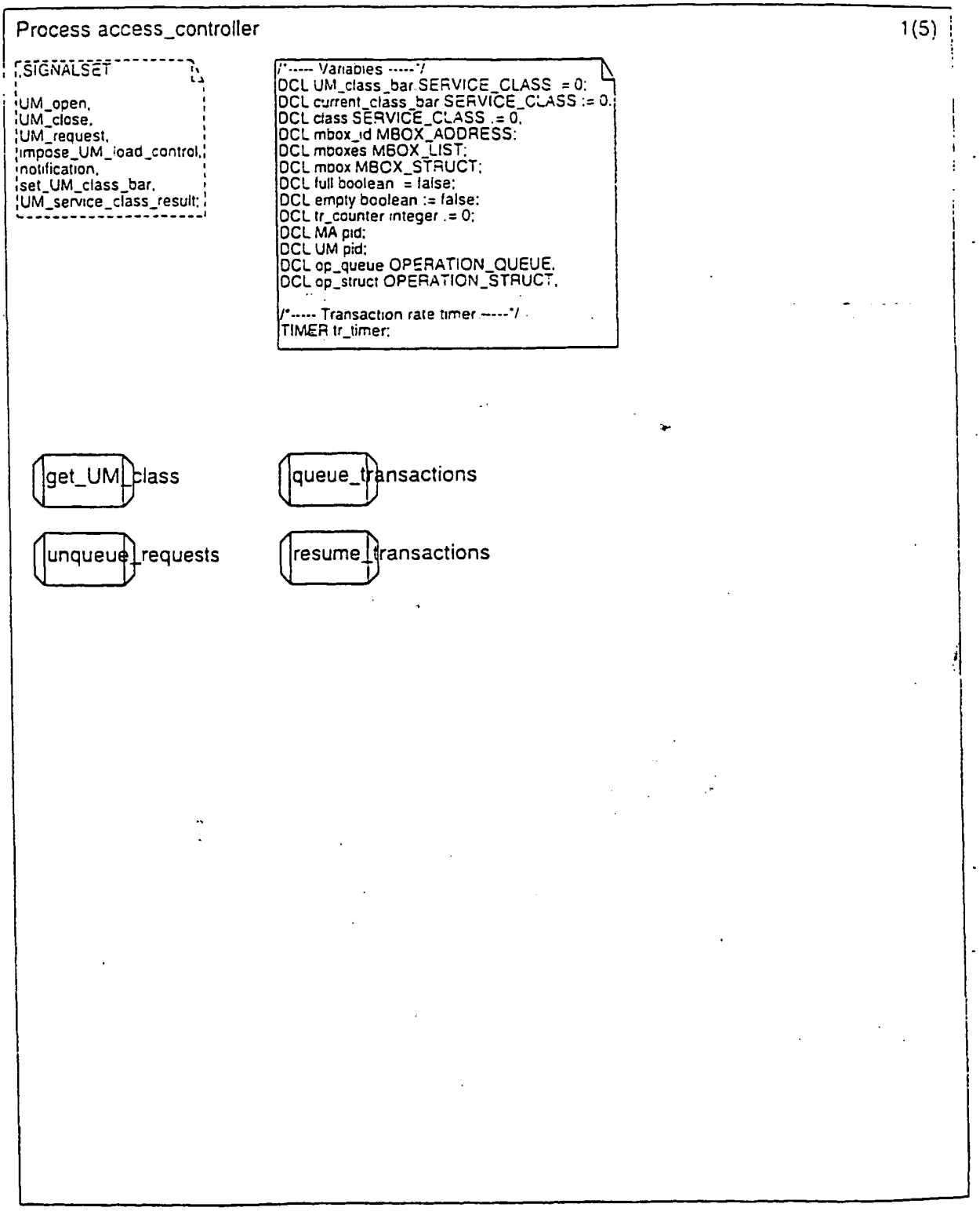

Process access_controller
1(5)
SIGNALSET
UM_open,
UM_close,
UM_request,
impose_UM_load_control,
notification,
set_UM_class_bar,
UM_service_class_result;
/*----- Variables -----*/
DCL UM_class_bar SERVICE_CLASS := 0;
DCL current_class_bar SERVICE_CLASS := 0;
DCL class SERVICE_CLASS := 0;
DCL mbox_id MBOX_ADDRESS;
DCL mboxes MBOX_LIST;
DCL mbox MBOX_STRUCT;
DCL full boolean := false;
DCL empty boolean := false;
DCL tr_counter integer := 0;
DCL MA pid;
DCL UM pid;
DCL op_queue OPERATION_QUEUE;
DCL op_struct OPERATION_STRUCT;
/*----- Transaction rate timer -----*/
TIMER tr_timer;
get_UM_class
queue_transactions
unqueue_requests
resume_transactions

Figure 31

Process access_controller 2(5)

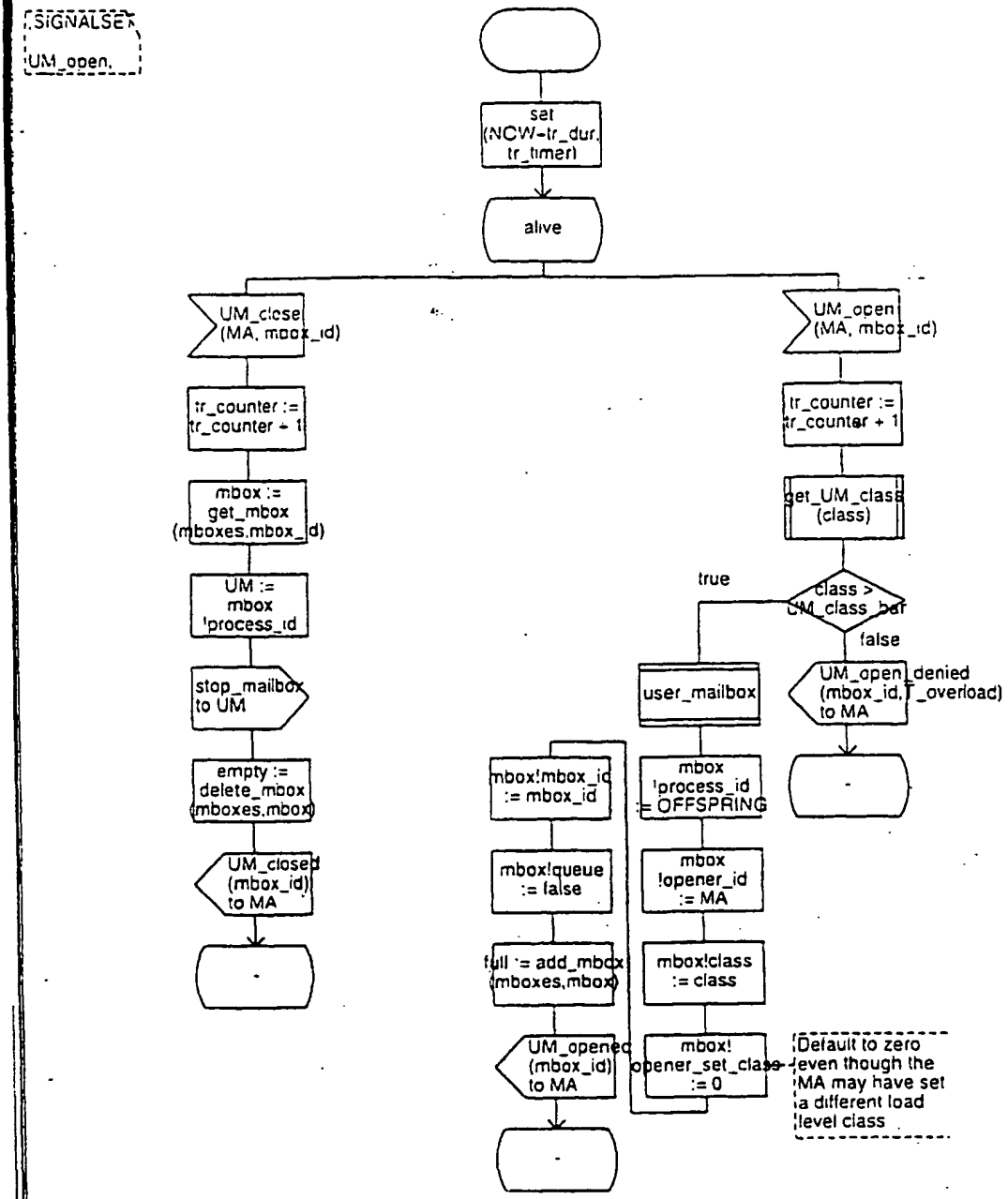


Figure 3m

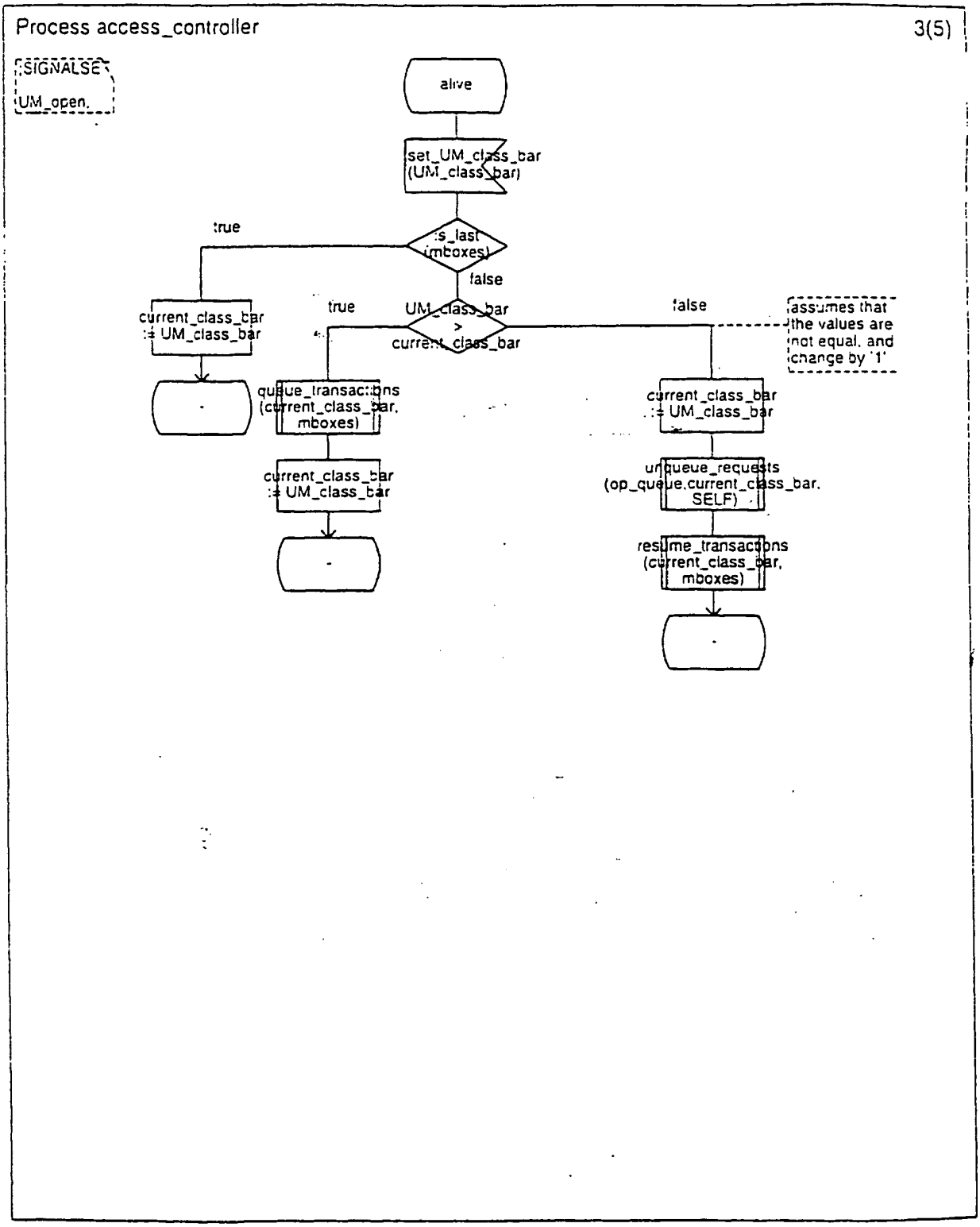

Process access_controller
3(5)
SIGNALSET
UM_open
alive
set_UM_class_bar
(UM_class_bar)
true
s_last
(mboxes)
false
current_class_bar
:= UM_class_bar
true
UM_class_bar
>
current_class_bar
false
assumes that the values are not equal, and change by '1'
queue_transactions
(current_class_bar, mboxes)
current_class_bar
:= UM_class_bar
current_class_bar
:= UM_class_bar
unqueue_requests
(op_queue,current_class_bar, SELF)
resume_transactions
(current_class_bar, mboxes)

Figure 3n

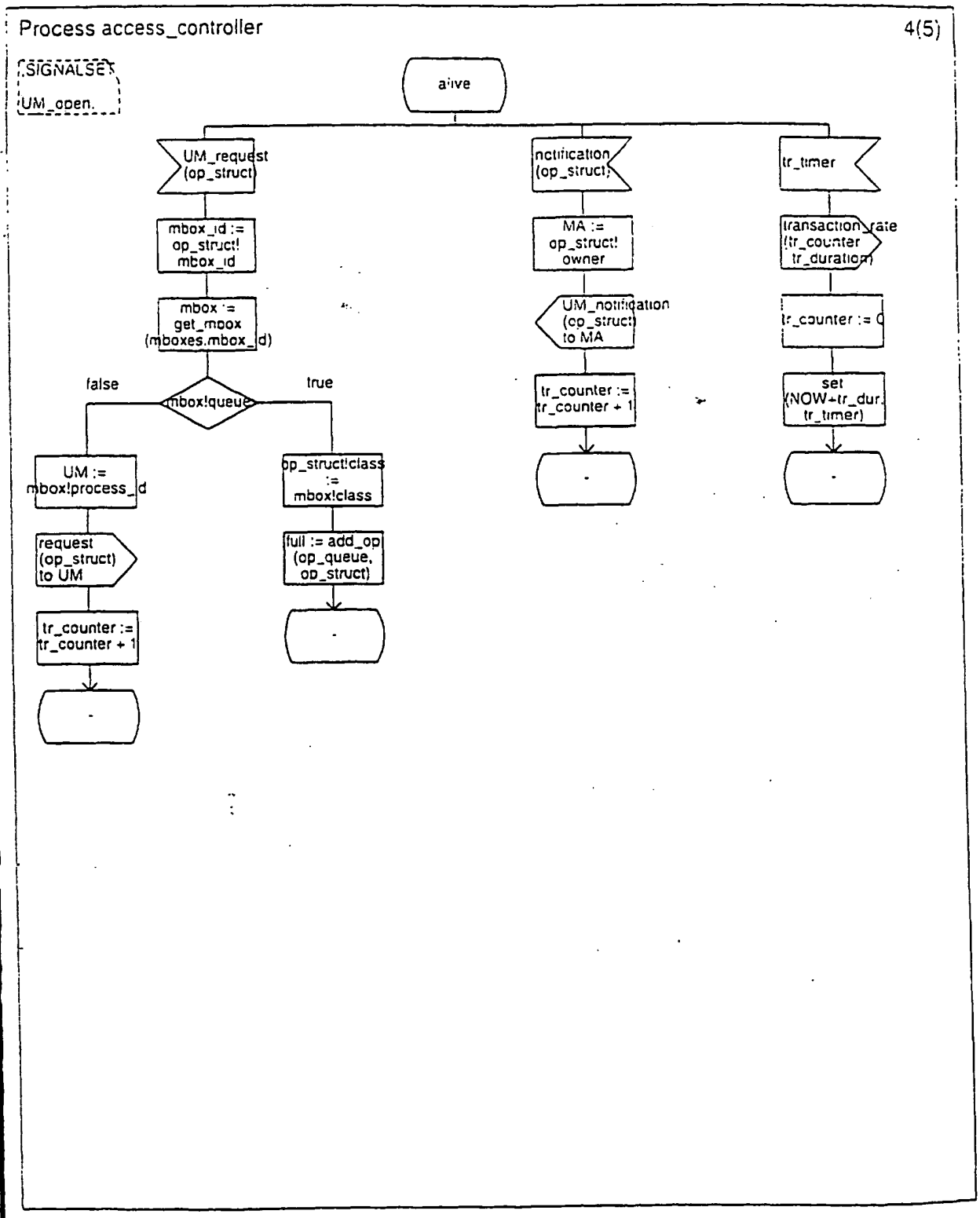

Process access_controller
4(5)
SIGNALSET
UM_open
alive
UM_request (op_struct)
notification (op_struct)
tr_timer
mbox_id := op_struct! mbox_id
MA := op_struct! owner
transaction_rate (tr_counter tr_duration)
mbox := get_mbox (mboxes.mbox_id)
UM_notification (op_struct) to MA
tr_counter := 0
false
mbox!queue
true
tr_counter := tr_counter + 1
set (NOW+tr_dur, tr_timer)
UM := mbox!process_id
op_struct!class := mbox!class
request (op_struct) to UM
full := add_op (op_queue, op_struct)
tr_counter := tr_counter + 1

Figure 3o

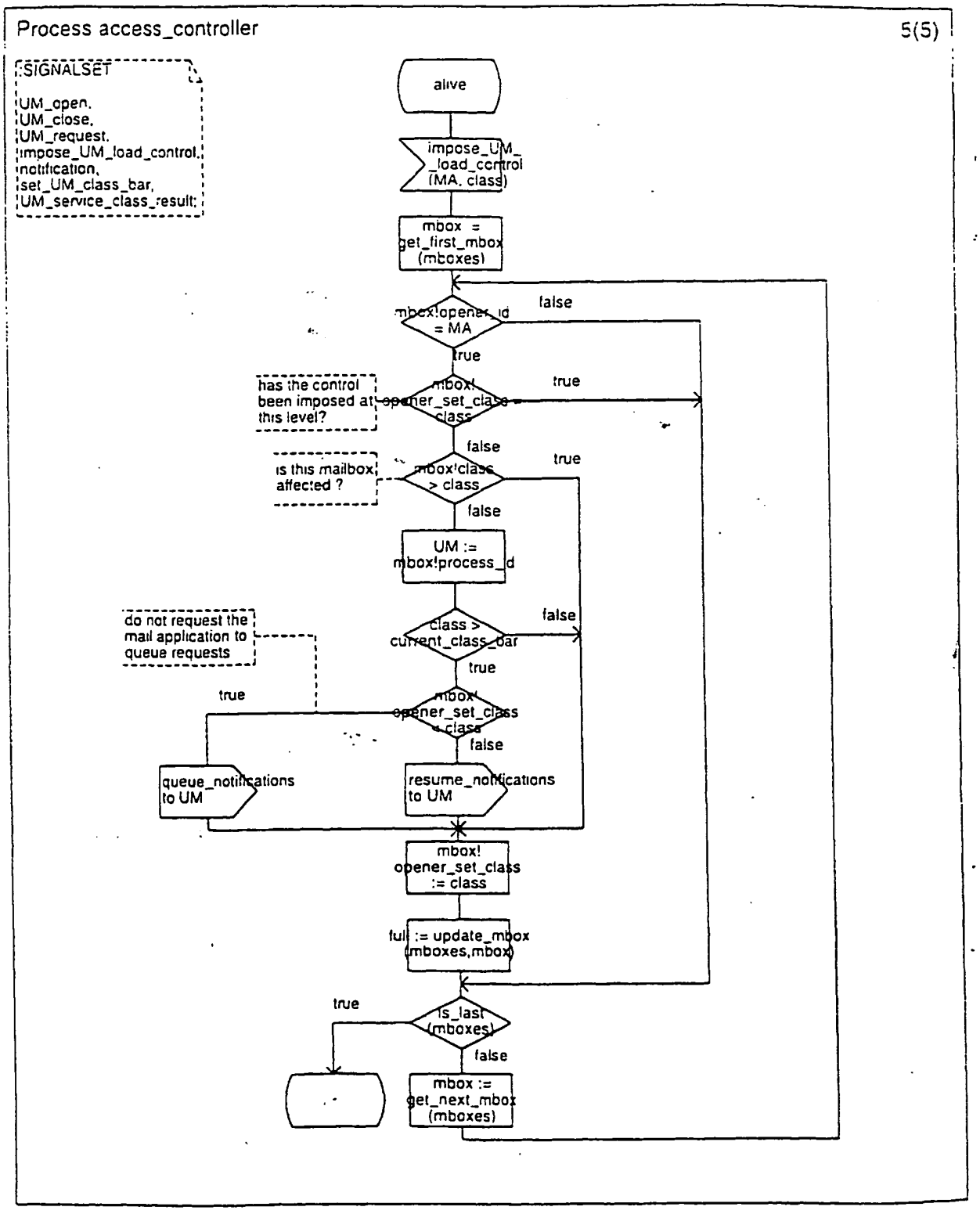

Process access_controller
5(5)
SIGNALSET
UM_open,
UM_close,
UM_request,
impose_UM_load_control,
notification,
set_UM_class_bar,
UM_service_class_result;
alive
impose_UM_load_control (MA, class)
mbox = get_first_mbox (mboxes)
mbox!opener_id = MA
false
true
has the control been imposed at this level?
mbox! opener_set_class = class
true
false
is this mailbox affected ?
mbox!class > class
true
false
UM := mbox!process_id
do not request the mail application to queue requests
class > current_class_bar
false
true
true
mbox! opener_set_class = class
false
queue_notifications to UM
resume_notifications to UM
mbox! opener_set_class := class
full := update_mbox (mboxes,mbox)
true
is_last (mboxes)
false
mbox := get_next_mbox (mboxes)

Figure 3p

Procedure get_UM_class

1(1)

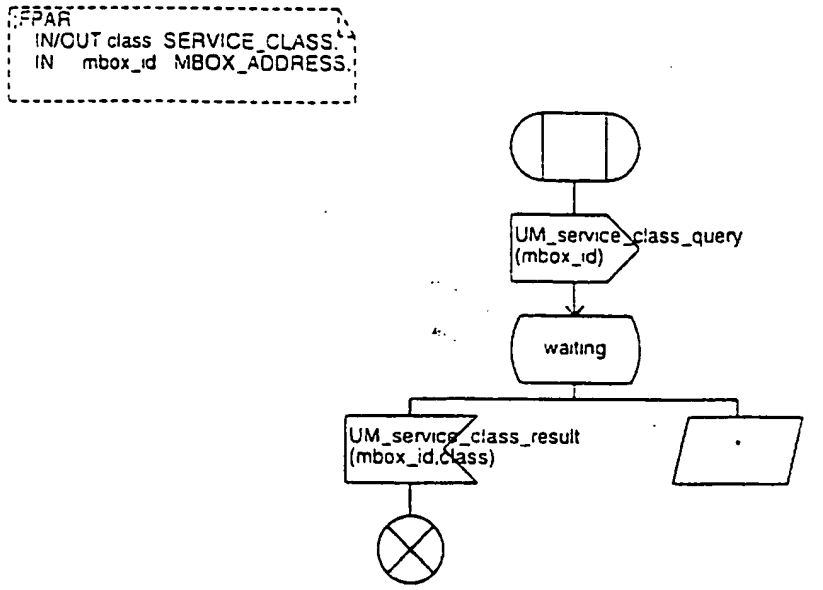


Figure 3q

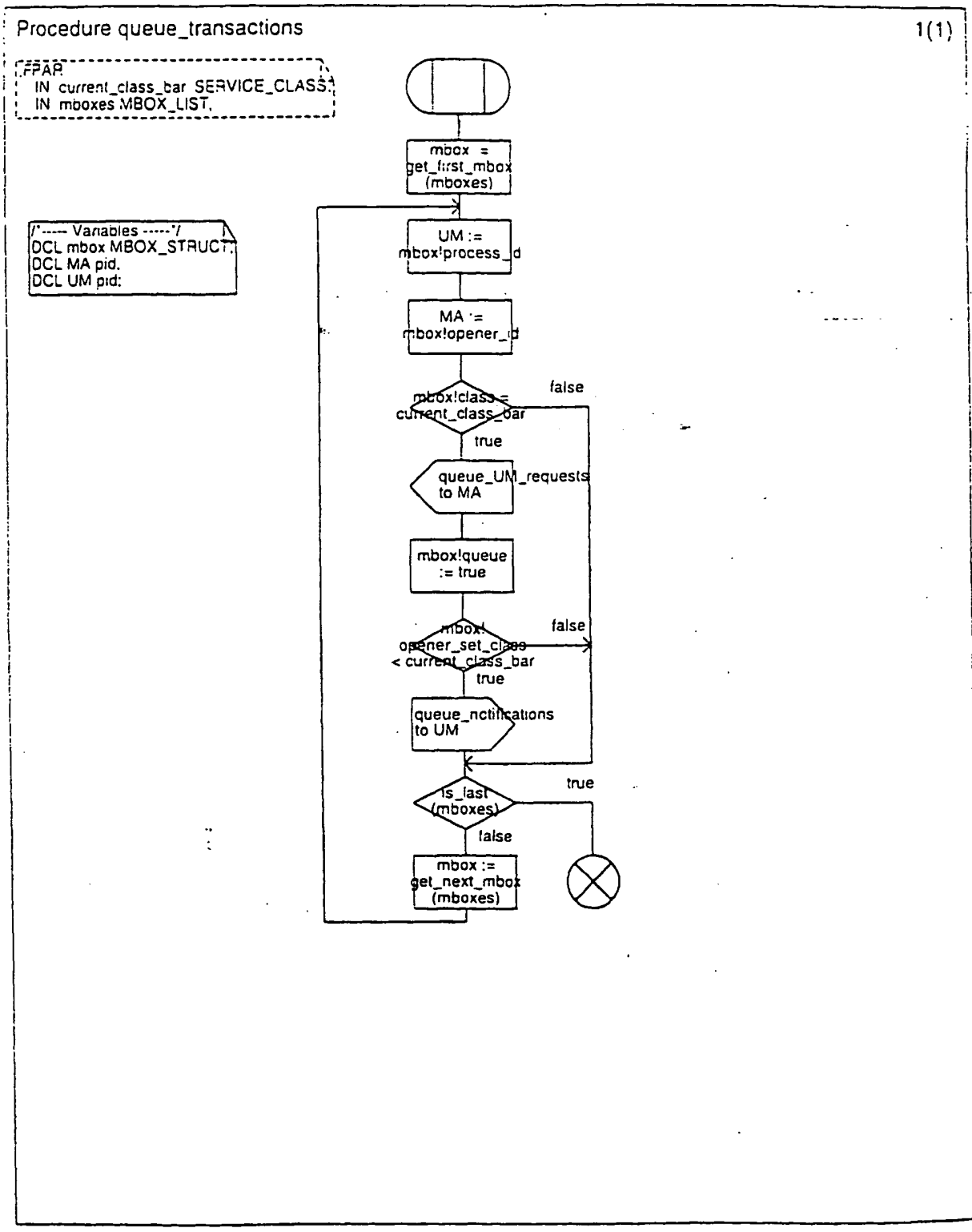

Procedure queue_transactions
1(1)
FPAR
IN current_class_bar SERVICE_CLASS,
IN mboxes MBOX_LIST,
/*----- Variables -----*/
DCL mbox MBOX_STRUCT,
DCL MA pid,
DCL UM pid;
mbox = get_first_mbox (mboxes)
UM := mbox!process_id
MA := mbox!opener_id
mbox!class = current_class_bar
false
true
queue_UM_requests to MA
mbox!queue := true
mbox! opener_set_class < current_class_bar
false
true
queue_notifications to UM
is_last (mboxes)
true
false
mbox := get_next_mbox (mboxes)

Figure 3r

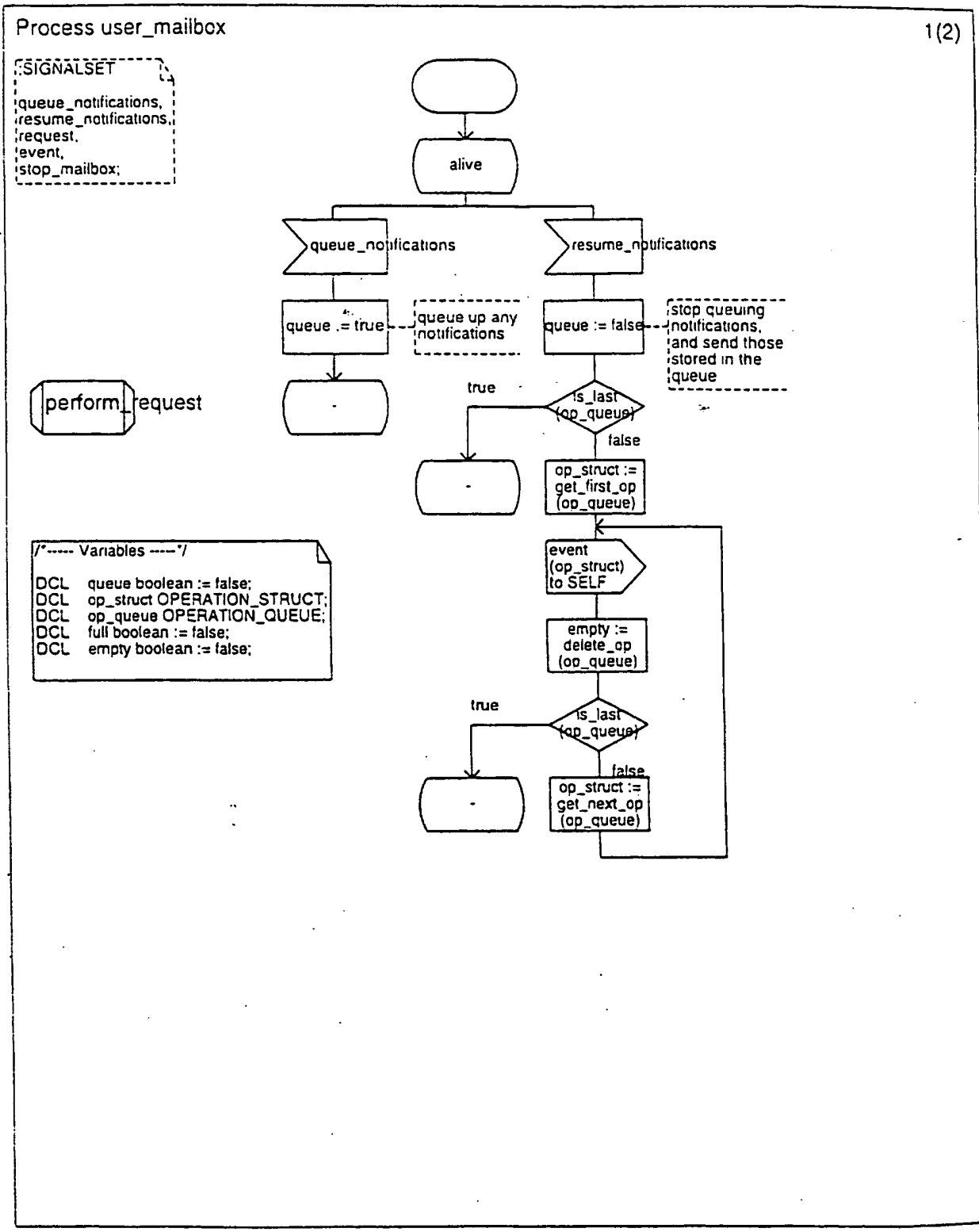

Process user_mailbox
1(2)
SIGNALSET
queue_notifications,
resume_notifications,
request,
event,
stop_mailbox;
alive
queue_notifications
resume_notifications
queue := true
queue up any notifications
queue := false
stop queuing notifications, and send those stored in the queue
perform request
true
is_last (op_queue)
false
op_struct := get_first_op (op_queue)
event (op_struct) to SELF
empty := delete_op (op_queue)
true
is_last (op_queue)
false
op_struct := get_next_op (op_queue)
/*----- Variables -----*/
DCL queue boolean := false;
DCL op_struct OPERATION_STRUCT;
DCL op_queue OPERATION_QUEUE;
DCL full boolean := false;
DCL empty boolean := false;

Figure 3w

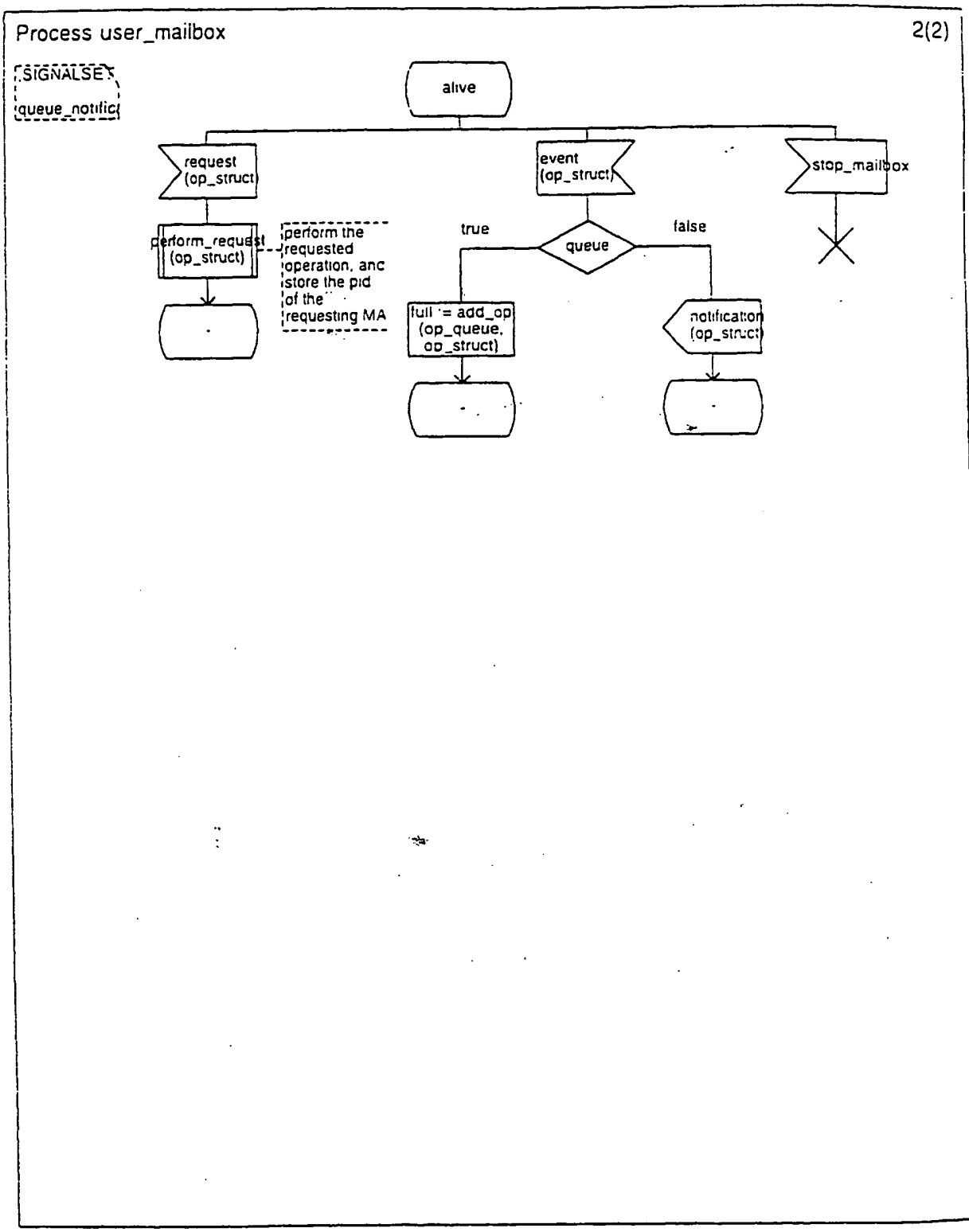

Process user_mailbox
2(2)
SIGNALSET
queue_notific
alive
request
(op_struct)
event
(op_struct)
stop_mailbox
perform_request
(op_struct)
perform the
requested
operation, and
store the pid
of the
requesting MA
true
queue
false
full := add_op
(op_queue,
op_struct)
notification
(op_struct)

Figure 3x

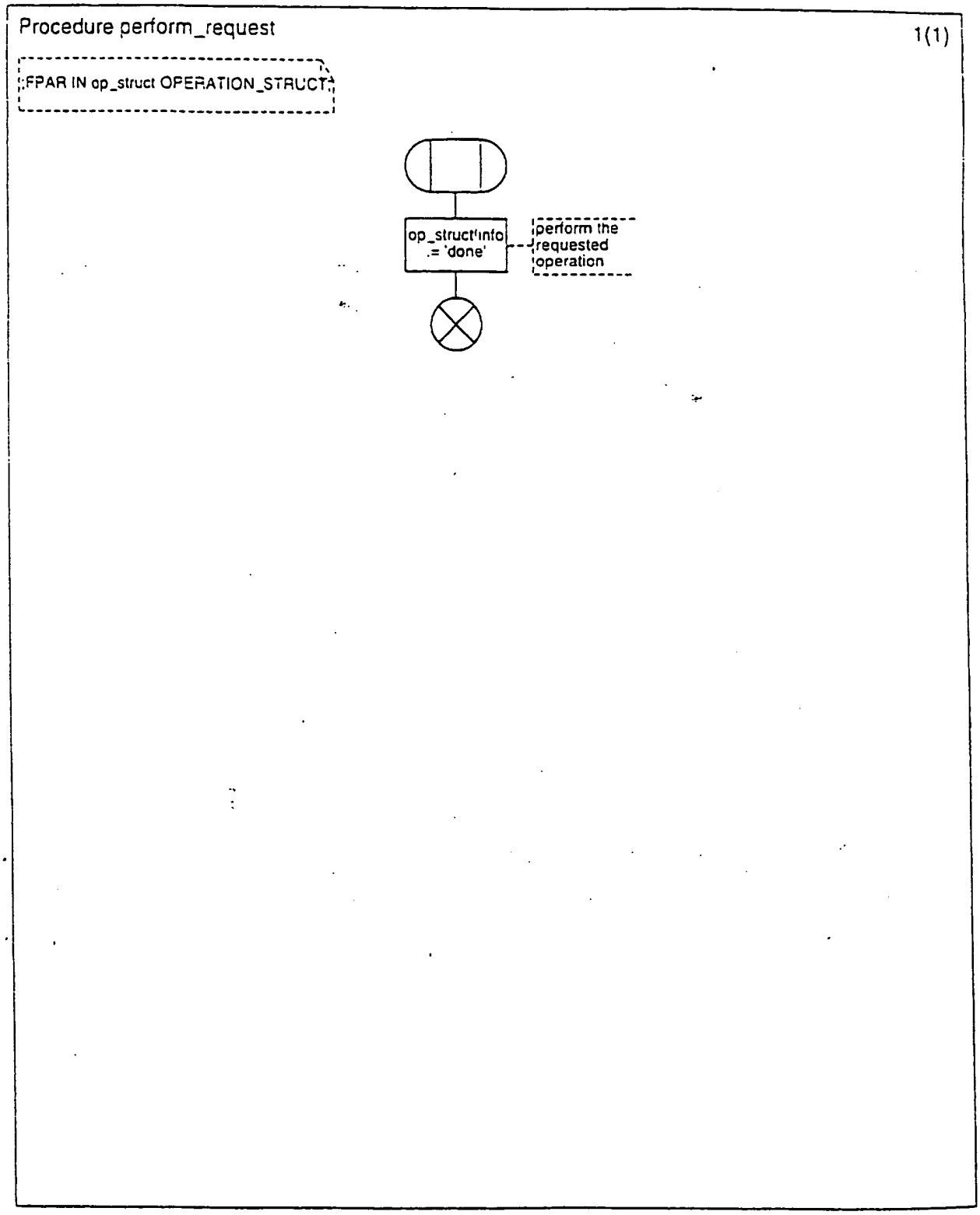

Procedure perform_request
1(1)
FPAR IN op_struct OPERATION_STRUCT
op_struct!info := 'done'
perform the requested operation

Figure 3y

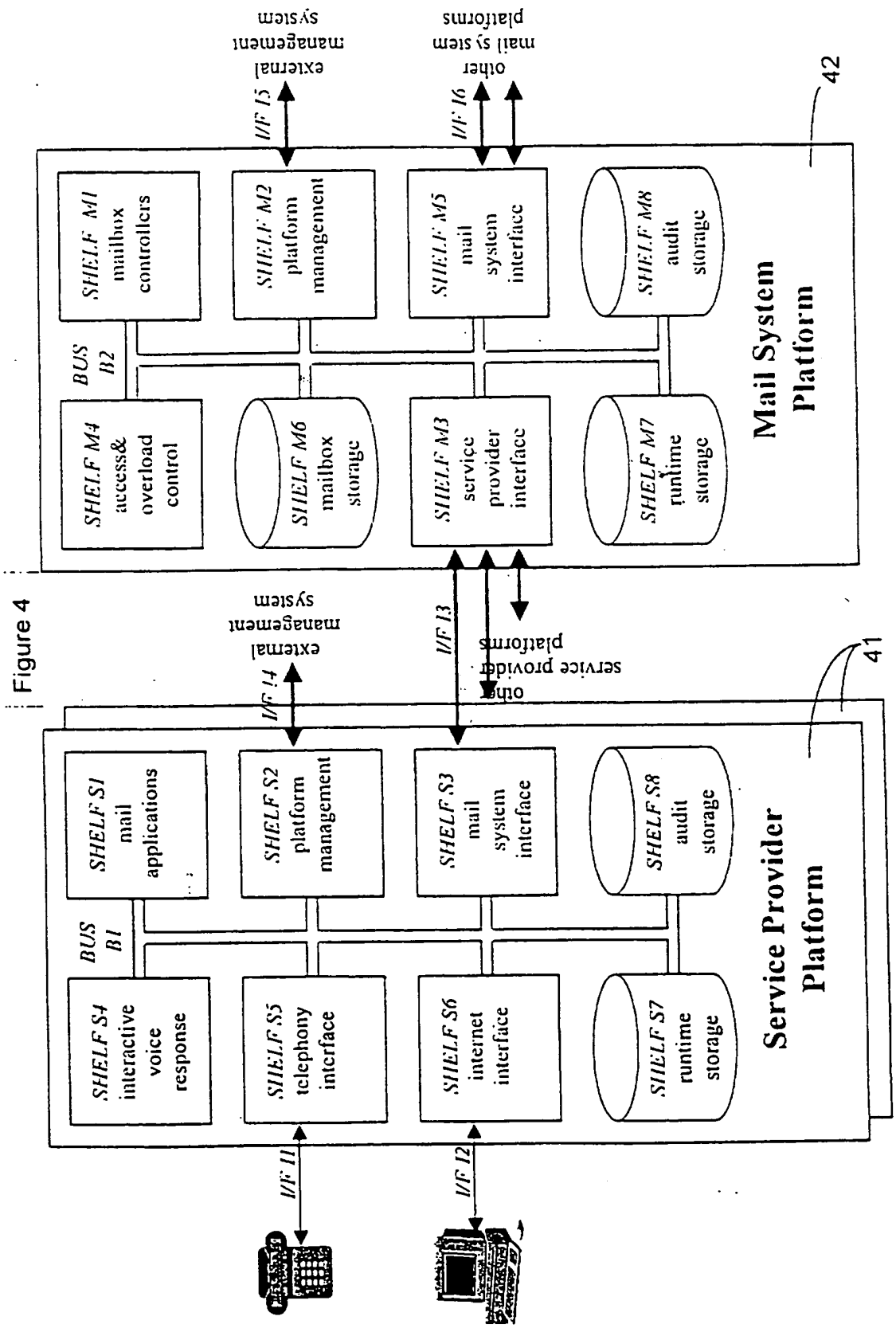

Service Provider Platform
41
SHELF S1 mail applications
SHELF S2 platform management
SHELF S3 mail system interface
SHELF S8 audit storage
BUS B1
SHELF S4 interactive voice response
SHELF S5 telephony interface
SHELF S6 internet interface
SHELF S7 runtime storage
I/F 11
I/F 12
I/F 13
I/F 14
external management system
other service provider platforms
Mail System Platform
42
SHELF M1 mailbox controllers
SHELF M2 platform management
SHELF M5 mail system interface
SHELF M8 audit storage
BUS B2
SHELF M4 access& overload control
SHELF M6 mailbox storage
SHELF M3 service provider interface
SHELF M7 runtime storage
I/F 15
I/F 16
external management system
other mail system platforms

Figure 4

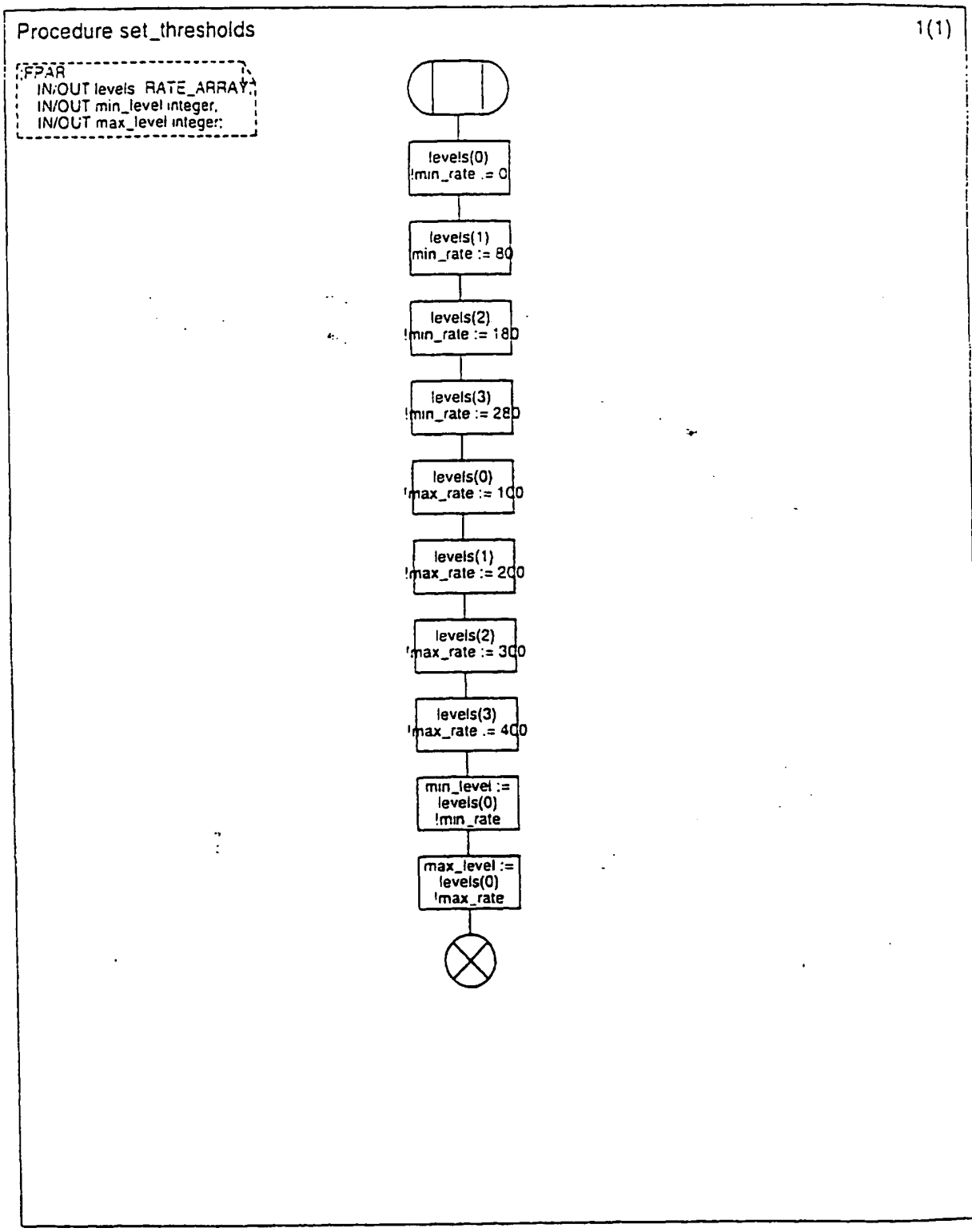

Procedure set_thresholds
1(1)
FPAR
IN/OUT levels RATE_ARRAY,
IN/OUT min_level integer,
IN/OUT max_level integer:
levels(0)
!min_rate := 0
levels(1)
min_rate := 80
levels(2)
!min_rate := 180
levels(3)
!min_rate := 280
levels(0)
!max_rate := 100
levels(1)
!max_rate := 200
levels(2)
!max_rate := 300
levels(3)
!max_rate := 400
min_level :=
levels(0)
!min_rate
max_level :=
levels(0)
!max_rate

Figure 3v

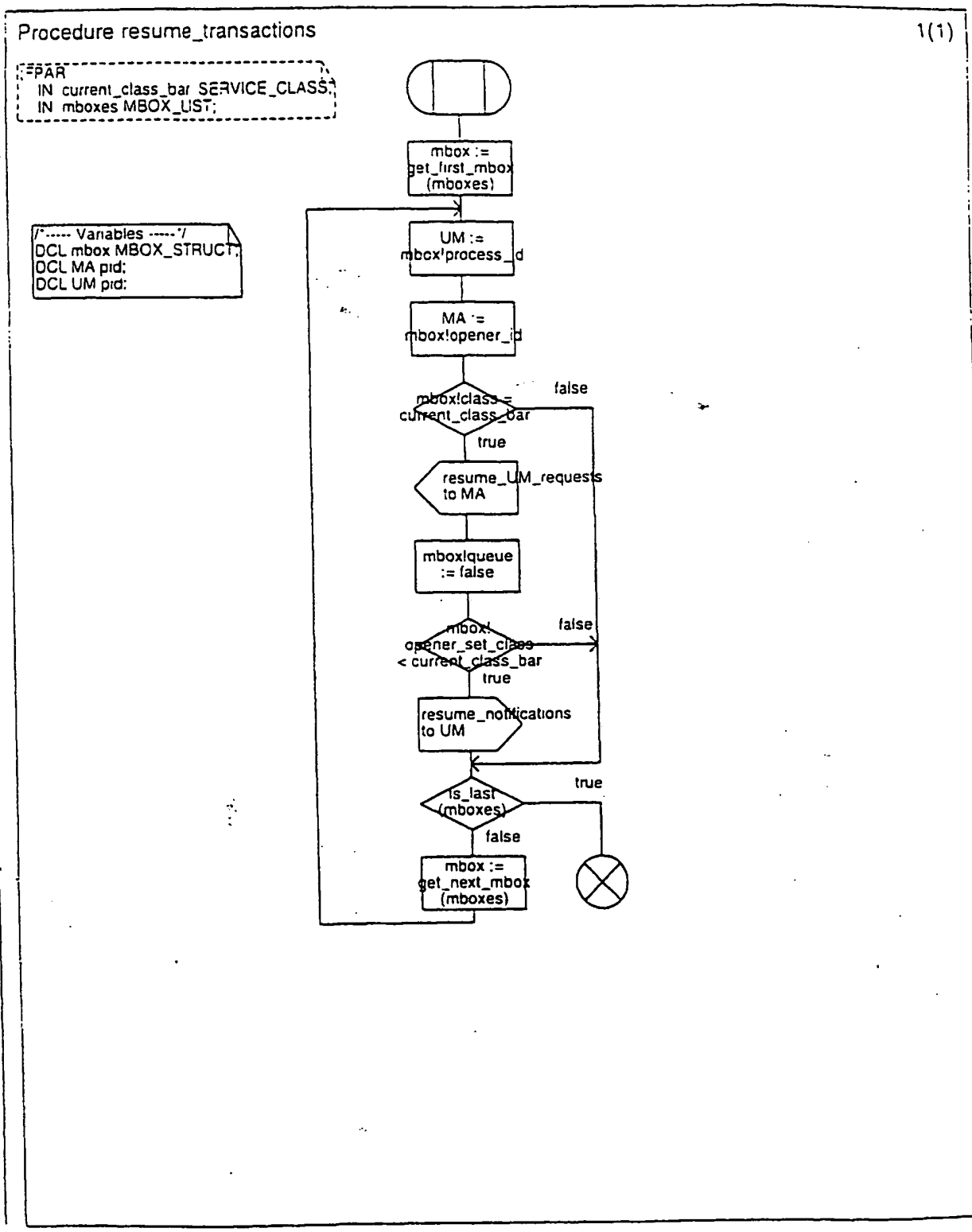

Procedure resume_transactions
1(1)
PAR
IN current_class_bar SERVICE_CLASS;
IN mboxes MBOX_LIST;
/*----- Variables -----*/
DCL mbox MBOX_STRUCT;
DCL MA pid;
DCL UM pid;
mbox := get_first_mbox (mboxes)
UM := mbox!process_id
MA := mbox!opener_id
mbox!class = current_class_bar
false
true
resume_UM_requests to MA
mbox!queue := false
mbox! opener_set_class < current_class_bar
false
true
resume_notifications to UM
is_last (mboxes)
true
false
mbox := get_next_mbox (mboxes)

Figure 3s